# WILL YOU STILL LOVE ME WHEN I'M 64

Peter Gerrard

# WILL YOU STILL LOVE ME WHEN I'M 64

Peter Gerrard

Duckworth

First published in 1985 by
Gerald Duckworth & Co. Ltd.
The Old Piano Factory
43 Gloucester Crescent, London NW1



ISBN 0 7156 1783 4

British Library Cataloguing in Publication Data
Gerrard, Peter
Will you still love me when I'm 64. — (Duckworth home computing)
1. Commodore 64 (Computer)
I. Title
001.64'04 QA76.8.C64

ISBN 0-7156-1783-4

Typeset by The Electronic Village, Richmond
from text stored on a Commodore 64
Printed in Great Britain by
Redwood Burn Ltd., Trowbridge
and bound by Pegasus Bookbinding, Melksham

# Contents


| | | |
|---|---|---|
| Preface | | 7 |
| 1. | Introduction to Music | 9 |
| 2. | The Theory of Sound | 15 |
| 3. | Musical Tables and Values | 21 |
| 4. | What the 64 Lets Us Do | 27 |
| 5. | In Which We Play a Tune | 36 |
| 6. | Three-Part Harmony | 42 |
| 7. | Getting Tuneful | 55 |
| 8. | Ring Modulation and Synchronisation | 67 |
| 9. | Filtering Techniques | 84 |
| 10. | A Synthesiser | 93 |
| 11. | Generating Sound Effects | 104 |
| 12. | Musical Games Programming | 115 |
| 13. | Musical Interrupts | 130 |
| 14. | Adding Musical Commands to Basic | 141 |
| 15. | Technical Overview | 164 |
| Index | | 185 |

# Preface

To the best of my knowledge there is no book devoted purely to sound on the 64, other than this one (don't all write in and tell me if there is!). I've assumed that the reader is reasonably competent in Basic, and may even have a little knowledge of machine code as well. Consequently, any machine code programs are usually presented with a Basic loader program as well, to help those of you who have not yet got to grips with both languages.

The book covers everything concerned with music on the 64, including ring modulation, synchronisation, adding musical commands to Basic, and producing interrupt-driven background music. There are around 20 major programs listed in the book. For those of you whose fingers aren't up to typing in reams and reams of program code, the lengthier programs are all available on tape from the publishers at a cost of £7.95.

And with that, happy programming!

A quick dedication, to Tony and Wendy, and all the bar staff at the Swan, in Ashford. Thanks, and may you never run out of bottled J.C.

P.G.

# 1
# Introduction to Music

How many of us have watched a skilful busker playing on a street corner, a soloist performing at the last night of the Proms, Jimi Hendrix making a guitar talk, Oscar Peterson making a piano dance, and said to ourselves 'I wish I could do that'?

In truth, most people have probably, at one time or another, wanted to play a musical instrument of some kind. Some have probably even tried to learn how to play the guitar, the piano, the violin, or one of the other popular instruments, but have given up in abject failure and taken to building up a large record collection instead.

With the advent of synthesisers, the ability to play music of a sort came a step nearer for a lot of people, since anyone can make a noise by turning a few buttons and twiddling a few dials. And to all the musicians who are convinced that synthesisers will be the death of music as we know it (and I know one or two: hello Owen!), give us lesser mortals a break and allow us to make a noise or two of our own. A talented musician will always coax more out of a synthesiser than an un-talented one: they've got an ear for music, whereas most of us haven't.

Then along came the home computer, and with it some of the most sophisticated musical electronics at an affordable price. One such computer is, of course, the Commodore 64, arguably the best musical computer at the moment. Also, as we all know only too well, it has one of the worst Basics of all time when it comes to playing music or producing graphics. With the vast number of two commands (PEEK and POKE) to enable us to communicate with the sonic and visual capabilities of the beast, many people must have decided that Commodore can't want us to get the best out of the machine and that it isn't worth the effort of remembering where all the memory locations are that need to be altered.

Well, I can't defend the language of the 64 as I too think it is diabolical.

However, a little persistence will produce some quite satisfactory results from the machine, and the purpose of this book is to explore to the full the musical performance of the Commodore 64.

## The scope of the book

When starting to compile a book such as this there are one or two questions that the author must ask himself. Principal among these is obviously what to include and what to leave out. Include too much and you end up with a book that makes War and Peace look like a single sheet of A4 paper. Include too little, and the resulting book will be about as popular as german measles, so a reasonable balance has to be found. I hope I've achieved that, since just about everything is covered somewhere in the book, although perhaps not in the detail that some people might like. Scores of chapters about ring modulation techniques might be of interest to some people, but don't really belong in a general book such as this one. On the other hand, leave the subject out altogether and we're missing one of the features that makes the 64 stand out from the rest, and so it is covered in Chapter 8.

Another problem is the level at which to pitch the book. Do you assume that everyone is a machine code genius who knows nothing about sound, or a budding Beethoven who couldn't program a computer if his life depended on it? As ever, the wisest route is the one down the middle of these two extremes. So I've assumed that the person who buys this book is reasonably interested in music, and has a fairly competent grasp of Basic programming without being able to write Easyscript in their sleep.

The machine code programs in this book only appear towards the end, and in case you (a) haven't got an assembler or (b) think an assembler is someone who works on a production line at British Leyland, all the machine code is presented with a Basic set of equivalent data statements. This way, anyone can who can type will be able to get the program running. If you understand machine code, you can follow the logic of the program. If you don't, you can still get the program to perform properly without knowing too much about how it works.

When learning a new subject, it is always a good idea to have something happening as you progress. Therefore, rather than just presenting you with a lot of theory that would be about as interesting as a slice of bread, most of the chapters in this book have at least one reasonably long program in them. Whether this is a utility, a learning program, or a simple game, depends on the chapter in question, but

there will be something that will enable you to see (and hear) what you're learning about.

### Topics covered

After this brief introduction, we're going to be taking a look at the background to music generally, with a little bit about the history of music, the development of various instruments, and the actual physics of sound: how a note is produced, and how that note is affected by various parameters. If physics is not your high point, don't worry: I got a grade 6 at O-level, and I understand what I'm on about.

Then, before diving into the theory and making the 64 produce all manner of weird and wonderful noises, any necessary tables will be dealt with in one chapter, which will allow them to act as a useful reference and keep them all in one place so that you don't have to tear around the book in order to discover that the frequency of the note middle C is 261.63 hertz.

A final diversion before we make some sounds is presented in Chapter 4, which talks about how the 64 performs as a musical instrument, and what parameters we have to play with in order to produce the desired result. Then, we're off into the realms of music, covering just one voice to begin with, then on to three-part harmony, building up popular tunes, looking at ring modulation and synchronisation, filtering techniques, and so on.

Most of this is covered in more or less detail in the programmer's reference guide. However, it is hoped not only that a fresh approach will be useful, but that the programs presented in these early chapters will act as educational tools for all levels of competence.

The last half of the book covers new ground, starting off with building the 64 up to be a true synthesiser (filtering, synchronisation of voices, background rhythms, glissando effects, and more). This leads on to sound effects and games programming with sound, before we end up with a look at how background rhythms are produced using interrupts, and how new commands can be added to Basic in order to make musical life a whole lot easier. Needless to say, there will be practical examples to cover all the topics under discussion.

The final chapter is for those of you who want to know how the marvellous 6581 chip works. The 6581, perhaps better known as the SID chip, is a custom-made Commodore chip, rather than an off-the-

shelf General Instruments chip which is what everyone else seems to use. The difference between the two chips goes a long way towards explaining why the 64 stands out above the rest.

## Some conventions

This is concerned more with the program listings than anything else, as the one topic that produces more complaints than anything else is the legibility of listings (and of course whether or not they work).

All the listings in this book were produced on an Epson FX-80 printer, operating at half its normal speed in double density print mode. To put it another way, the listings are twice as clear as they would normally be. They are also printed out to a line width of fifty columns, with anything longer wrapping around on to the next line of print. Be careful when typing in lines that wrap around this way. Don't be tempted to include spaces that might not be there, just carry on typing.

A large number of the programs have lines that, on the face of it, are longer than 80 columns. This is because I am forever using ? instead of PRINT, T shifted H instead of THEN, P shifted O instead of POKE, and so on. If you come across a line that looks inordinately long, you will have to use these abbreviations in order to get it to fit.

A word of warning. The 64 has a curious bug when it comes to entering very long lines. If, for some reason, you're at the bottom of the screen and realise that you've attempted to go on to the third line of a program line (in other words, a line longer than 80 columns) do not try to use the delete key to take yourself back onto the second line. The 64 will very probably hang, and there is no reliable way to get it back again, which means that your precious program will be lost. One possible solution is to press the play (or sometimes both play and record) button on your tape deck, but this does not work all the time, so take care.

The Epson printer cannot handle Commodore upper and lower case, and so everything appears at first glance to be in upper case only. This is not so, as a closer look will reveal that some of the letters are in *italics*. Enter any *italicised* letters using the shift key, and all the rest without the shift key.

The Epson (without a great deal of time spent programming user-defined characters and writing a special program to trap Commodore ones and convert them) cannot handle Commodore's control codes for things like cursor up, going into black print mode, and so on.

Accordingly, all these have been stripped from the program and replaced with various mnemonics. These are as follows:

| | | | |
|---|---|---|---|
| HOME | - Home key | CLR | - Shift and home key |
| DEL | - Delete key | INST | - Shift and delete key |
| RVS | - CTRL key and 9 | OFF | - CTRL key and 0 |
| CU | - Cursor up | CD | - Cursor down |
| CL | - Cursor left | CR | - Cursor right |
| BLK | - CTRL key and 1 | WHT | - CTRL key and 2 |
| RED | - CTRL key and 3 | CYN | - CTRL key and 4 |
| PUR | - CTRL key and 5 | GRN | - CTRL key and 6 |
| BLU | - CTRL key and 7 | YEL | - CTRL key and 8 |
| BRN | - CBM key and 2 | SP | - Space bar |

To enter more than one command at a time, commas and numbers are used, as in the following example:

[CLR,RVS,2CD,5CR]

This means 'press the shift and home keys, then the CTRL and 9 key, then the cursor down key twice and the cursor right key five times'. Do not enter any commas, they are only there as separators to make things easier to read.

Needless to say, the Epson is no great shakes when it comes to printing out Commodore graphical characters either. Why use an Epson at all then? Well, the image is clear, and have you ever tried entering a listing from a magazine that has used the standard Commodore dot matrix printer? It is virtually impossible to read.

Accordingly, any graphics characters that needed to be printed have also been replaced by mnemonics. These are usually of the form 'CBMM', which means 'press the CBM key in conjunction with the M key', or 'shift*', which means 'press the shift key in conjunction with the * key'. On some occasions we differ from this, but if there are any changes they are explained in REM statements.

One last point about the listings. If you can't face typing them in, all the longer ones are available on cassette from the publishers, at a cost of £7.95. This contains some nineteen programs in all (plus a couple of short 'driver' programs), which will save a lot of wear and tear on fingers and tempers.

# Conclusion

By the end of this book you should be in a position to produce your own musical programs, your own background rhythms and tunes (and please, please, always give a person the option of turning these off!), and be able to add your own musical commands to Basic.

Before we begin our musical voyage of discovery, we'll take a look at the history of music and the theory of sound.

# 2
# The Theory of Sound

## Introduction

While it is not essential to understand how the physics of sound operates (just as it isn't essential to know why POKE54272 + 24,15 sets the volume control of the 64 to its maximum level), it will make life easier as well as making a lot of what we'll be talking about make a great deal more sense.

In the same way, it is not really necessary to know how various musical instruments developed (just as you don't need to know that an attack/decay rate of 96, a sustain/release rate of 0 and a sawtooth wavelength will make one of the voices of the 64 sound rather like a trumpet), but again it will help to give you a more thorough understanding of how different results and effects are achieved.

There are five main families of musical instruments: string (e.g. a banjo), brass (e.g. a trumpet), woodwind (e.g. a flute), percussion (e.g. drums) and keyboard (e.g. a piano), although some, like the harp for instance, cannot be classified quite as easily as that. Arguably the most important instrument is the human voice, but since the Commodore 64 is not very adept at impersonating it we'll leave that topic to one side.

### String instruments

The most common string instrument today is probably the guitar, and as you know guitars come in all shapes and sizes. Other members of this family include things like the fiddle, the violin (which is basically a fiddle played in a different style), the banjo, and many more. Their ancestors were lutes, viols, rebecs, etc. Lesser-known string instruments include such diverse ones as the autoharp and the zither (did you know that 'Tales from the Vienna Woods' (Johann Strauss) includes a zither solo?)

The first violin was designed in the late sixteenth century by one Gasparo da Salo. Since then the violin has changed very little. Everyone has heard of da Salo's most famous pupil, Antonio Stradivari. He made around eleven hundred violins, of which about six hundred are still in existence. Manage to find one of these, and you've made yourself a fortune! How much will a Commodore 64 cost in four hundred years time, I wonder?

### Brass instruments

It is known that early man could punch a hole in the side of an animal horn and by blowing across that hole, produce a sound that could travel short distances and was useful for communicating. That method of blowing over (or indeed into) a hole is still the method used to play all the brass instruments, although not many orchestras feature animal horns as one of their leading solo performances, thank goodness.

To improve the sound produced, and thus send signals over even longer distances, something larger was obviously needed. This led to the development of massive instruments like the Swiss Alpine horn, or Alphorn, some of which are twelve feet or more in length.

Metals were soon put to use to make musical instruments, in the shape of short tubes, long tubes, cone-shaped tubes and straight tubes. Holes were made in a tube in various places, which could be covered or uncovered to change the notes produced by blowing into the tube.

With the invention of a valve mechanism in 1815, which allowed leading holes to extra pieces of tubing to be opened and closed, we begin to see instruments that we're familiar with today. Before that, trumpets had been used by various monarchs to play pageants and fanfares, and other diverse pieces of brass had also appeared.

Nowadays we have such different instruments as sousaphones (named after John Philip Sousa, a bandmaster and composer), tubas, euphoniums and so on, and the Commodore 64 can produce a passable impersonation of most of these, as we shall see.

### Woodwind instruments

The modern woodwind instruments with which we are now so familiar also have a long history. The ancient pipes of Pan, the Greek god of forest, flocks and shepherds, consisted of several pipes of different

lengths bundled together. When it was discovered that the work of all these different pipes could be achieved by just one pipe with holes cut into it at suitable intervals, the type of instrument that is common today began to appear.

Paintings from ancient Egypt and Greece depict the forerunners of what we now know as the flute. This began to resemble the modern instrument sometime in the Middle Ages, when it was discovered that by blowing across the top of the notch (rather liking blowing across the top of a bottle) a very pleasing tone could be produced. A tone, moreover, that was easy to control.

Other instruments in this family include the clarinet and the saxophone, although some will argue that the latter instrument belongs with the brass family. The saxophone was invented in 1842 by one Adolphe Sax as a metal equivalent of other woodwind instruments. Some musicans can now play two saxophones at the same time, albeit with great difficulty, but using the 64 we can easily 'play' three at a time: a triumph for computers, perhaps?

Finally we have the double-reed family of instruments, consisting of the oboe, the English horn (which isn't English and isn't a horn!), the bassoon and the contrabassoon, the lowest-sounding instrument commonly used in an orchestra.

### Percussion instruments

There seems to be something about percussion instruments that appeals to children of all ages, as any parent will know. Give a child something to hit that makes a noise (preferably not another child) and you'll have a happy little person.

In this category come such devices as the snare drum, timpani, bass drum, gong, triangle, tambourine, and so on.Looking more like keyboard instruments but grouped into this section because of the way they are played, we also have vibraphones, xylophones, marimbas, glokenspiels, and others. Trying to impersonate the sounds made by these instruments can be an interesting exercise for the musical 64 owner.

### Keyboard instruments

Possibly the easiest of all musical instruments to emulate on the 64

are the keyboard ones, such as the piano and the organ. Technically the harp comes into this section as well, since it is basically the 'innards' of a piano standing on its side.

We are all now familiar with the sound and sight of the modern synthesiser, which has a piano keyboard. Some people take synthesisers a bit too far, and think that they can emulate any instrument under the sun. Indeed, there is a story that, during one recording session, one half of the rock group 10CC was keen to use a synthesiser for everything, including playing the bass guitar. This upset the bass guitarist of the band considerably, and the resulting row was enough to split up the band a short while later.

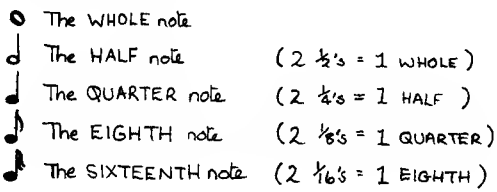
We won't be having any such rows, although we will, later on, be looking at ways of impersonating various musical instruments. As with all things musical connected with computers, the key to getting the best sound comes with experimentation. When you consider that, by using the pulse waveform, we have the option of using any one of 255 times 255 pulse widths, of having 255 times 255 different envelope shapes, and playing any one of 255 by 255 different notes (some not as musical as others), it would be a brave man who would say that he knew what every combination sounded like.

Before giving you a few tables and getting our hands on a 64 keyboard, we'll round off this chapter with some musical definitions and a discussion of the actual physics of sound.

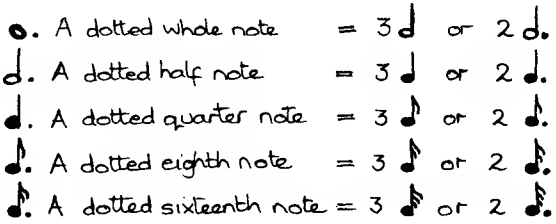
## Musical notation

Fortunately for musicians and computer people alike, there is a great deal of similarity between the binary system as used by computers and the standard musical notation used to determine how long a note will be played or a pause will be held.

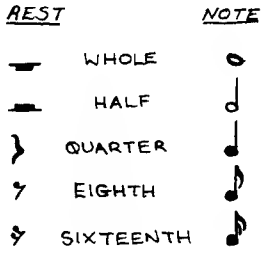
Starting with a standard value of one 'beat', for a whole note, this is divided up into halves, quarters, eighths, sixteenths and finally a thirty-second note, like this:



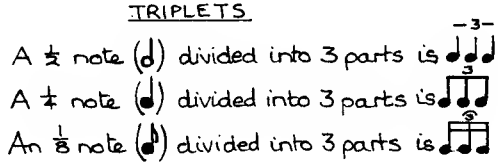
However, life has a way of getting complicated in even the simplest of settings, and there are other notes that we have to consider: the dotted notes. These are half as long again as their un-dotted companions:




Pauses, or rests, follow similar rules, and the symbols used to denote these look something like this:




And just to complicate the scene even further, it is sometimes deemed necessary to divide musical notes up into groups of three, referred to as triplets, like this:




Now we know what all the notes look like, how do we go about producing notes on the 64? The next chapter contains all the values needed for playing any note of the many that the computer is capable of, but it's interesting to discover how those values are arrived at. For that, we need to delve into a little bit of physics.

# The physics of sound

A note is just a series of ripples in the air, rather like the ripples on a pond's surface after you've lobbed a pebble into it. The frequency of a note depends on the distance between each ripple and the speed at which they're travelling. If it takes X seconds for two successive peaks to pass the same spot, then the frequency of the note is 1/X hertz.

Having found our frequency (say 261.63 hertz, or cycles per second), we then need a little bit of mathematics in order to arrive at the values we need to POKE into the 64. If we call this frequency FQ, then the first part of our series of equations becomes:

F = INT(FQ/0.05961)

giving us a value for F. In order to find the high-order value (FH) for the frequency (don't worry, I'll explain this in Chapter 4), we then need:

FH = INT(F/256)

To find the low order value, FL, we need to repeat the above equation without taking the INTeger value of it. What we're interested in this time is the value after the decimal point: say, 0.9856. Our equation for finding FL becomes:

FL = 256-256*0.9856

or whatever the decimal value worked out to be. Using these equations we could find out the high order and low order values for any note we like. For convenience, these are grouped together in one of the tables in Chapter 3. You may disagree with some of the values given (as does our synthesiser program later on in Chapter 10), so don't feel that you always have to use those values. A slight change will sometimes be required in order to make the note 'feel' exactly right.

# 3
# Musical Tables and Values

This collection of tables and musical values should enable you to play any note on the Commodore 64. They're all grouped together here in order to make it easy to find them, rather diving all over the book and having the pages scattering to the four corners of your room.

## Equal tempered musical scale

| MUSICAL NDTE | | FREQ (Hz) | DSC Fn (DECIMAL) | DSC Fn (HEX) | MUSICAL NDTE | | FREQ (Hz) | DSC Fn (DECIMAL) | DSC Fn (HEX) |
|---|---|---|---|---|---|---|---|---|---|
| 0 | C0 | 16.35 | 274 | 0112 | 48 | C4 | 261.63 | 4389 | 1125 |
| 1 | C0# | 17.32 | 291 | 0123 | 49 | C4# | 277.18 | 4650 | 122A |
| 2 | D0 | 18.35 | 308 | 0134 | 50 | D4 | 293.66 | 4927 | 133F |
| 3 | D0# | 19.44 | 326 | 0146 | 51 | D4# | 311.13 | 5220 | 1464 |
| 4 | E0 | 20.60 | 346 | 015A | 52 | E4 | 329.63 | 5530 | 159A |
| 5 | F0 | 21.83 | 366 | 016E | 53 | F4 | 349.23 | 5859 | 16E3 |
| 6 | F0# | 23.12 | 388 | 0184 | 54 | F4# | 370.00 | 6207 | 183F |
| 7 | G0 | 24.50 | 411 | 0188 | 55 | G4 | 392.00 | 6577 | 1981 |
| 8 | G0# | 25.96 | 435 | 01B3 | 56 | G4# | 415.30 | 6968 | 1B38 |
| 9 | A0 | 27.50 | 461 | 01CD | 57 | A4 | 440.00 | 7382 | 1CD6 |
| 10 | A0# | 29.14 | 489 | 01E9 | 58 | A4# | 466.16 | 7821 | 1E8D |
| 11 | B0 | 30.87 | 518 | 0206 | 59 | B4 | 493.88 | 8286 | 205E |
| 12 | C1 | 32.70 | 549 | 0225 | 60 | C5 | 523.25 | 8779 | 224B |
| 13 | C1# | 34.65 | 581 | 0245 | 61 | C5# | 554.37 | 9301 | 2455 |
| 14 | D1 | 36.71 | 616 | 0268 | 62 | D5 | 587.33 | 9854 | 267E |
| 15 | D1# | 38.89 | 652 | 028C | 63 | D5# | 622.25 | 10440 | 28C8 |
| 16 | E1 | 41.20 | 691 | 02B3 | 64 | E5 | 659.26 | 11060 | 2834 |
| 17 | F1 | 43.65 | 732 | 02DC | 65 | F5 | 698.46 | 11718 | 2DC6 |
| 18 | F1# | 46.25 | 776 | 0308 | 66 | F5# | 740.00 | 12415 | 307F |
| 19 | G1 | 49.00 | 822 | 0336 | 67 | G5 | 783.99 | 13153 | 3361 |
| 20 | G1# | 51.91 | 871 | 0367 | 68 | G5# | 830.61 | 13935 | 366F |
| 21 | A1 | 55.00 | 923 | 0398 | 69 | A5 | 880.00 | 14764 | 39AC |
| 22 | A1# | 58.27 | 978 | 03D2 | 70 | A5# | 932.33 | 15642 | 3D1A |
| 23 | B1 | 61.74 | 1036 | 040C | 71 | B5 | 987.77 | 16572 | 408C |
| 24 | C2 | 65.41 | 1097 | 0449 | 72 | C6 | 1046.50 | 17557 | 4495 |
| 25 | C2# | 69.30 | 1163 | 04BB | 73 | C6# | 1108.73 | 18601 | 48A9 |
| 26 | D2 | 73.42 | 1232 | 04D0 | 74 | D6 | 1174.66 | 19708 | 4CFC |
| 27 | D2# | 77.78 | 1305 | 0519 | 75 | D6# | 1244.51 | 20897 | 51BF |
| 28 | E2 | 82.41 | 1383 | 0567 | 76 | E6 | 1318.51 | 22121 | 5669 |
| 29 | F2 | 87.31 | 1465 | 05B9 | 77 | F6 | 1396.91 | 23436 | 5B8C |
| 30 | F2# | 92.50 | 1552 | 0610 | 78 | F6# | 1479.98 | 24830 | 60FE |
| 31 | G2 | 98.00 | 1644 | 066C | 79 | G6 | 1567.98 | 26306 | 66C2 |
| 32 | G2# | 103.83 | 1742 | 06CE | 80 | G6# | 1661.22 | 27871 | 6CDF |
| 33 | A2 | 110.00 | 1845 | 0735 | 81 | A6 | 1760.00 | 29528 | 735B |
| 34 | A2# | 116.54 | 1955 | 07A3 | 82 | A6# | 1864.65 | 31284 | 7A34 |
| 35 | B2 | 123.47 | 2071 | 0817 | 83 | B6 | 1975.53 | 33144 | 8178 |
| 36 | C3 | 130.81 | 2195 | 0893 | 84 | C7 | 2093.00 | 35115 | 892B |
| 37 | C3# | 138.59 | 2325 | 0915 | 85 | C7# | 2217.46 | 37203 | 9153 |
| 38 | D3 | 146.83 | 2463 | 099F | 86 | D7 | 2349.32 | 39415 | 99F7 |
| 39 | D3# | 155.56 | 2610 | 0A32 | 87 | D7# | 2489.01 | 41759 | A31F |
| 40 | E3 | 164.81 | 2765 | 0ACD | 88 | E7 | 2637.02 | 44242 | ACD2 |
| 41 | F3 | 174.61 | 2930 | 0B72 | 89 | F7 | 2793.83 | 46873 | B719 |
| 42 | F3# | 185.00 | 3104 | 0C20 | 90 | F7# | 2959.95 | 49660 | C1FC |
| 43 | G3 | 196.00 | 32BB | 0CDB | 91 | G7 | 3135.96 | 52613 | CDB5 |
| 44 | G3# | 207.65 | 34B4 | 0D9C | 92 | G7# | 3322.44 | 55741 | D98D |
| 45 | A3 | 220.00 | 3691 | 0E6B | 93 | A7 | 3520.00 | 59056 | E6B0 |
| 46 | A3# | 233.08 | 3910 | 0F46 | 94 | A7# | 3729.31 | 62567 | F467 |
| 47 | B3 | 246.94 | 4143 | 102F | 95 | B7 | 3951.06 | *66288 | *1F2F0 |

# Musical values

Here middle C has been denoted as C-4.

| Note | Note–Octave | Hi Freq | Low Freq |
|---|---|---|---|
| 0 | C–0 | 1 | 18 |
| 1 | C#–0 | 1 | 35 |
| 2 | D–0 | 1 | 52 |
| 3 | D#–0 | 1 | 70 |
| 4 | E–0 | 1 | 90 |
| 5 | F–0 | 1 | 110 |
| 6 | F#–0 | 1 | 132 |
| 7 | G–0 | 1 | 155 |
| 8 | G#–0 | 1 | 179 |
| 9 | A–0 | 1 | 205 |
| 10 | A#–0 | 1 | 233 |
| 11 | B–0 | 2 | 6 |
| 12 | C–1 | 2 | 37 |
| 13 | C#–1 | 2 | 69 |
| 14 | D–1 | 2 | 104 |
| 15 | D#–1 | 2 | 140 |
| 16 | E–1 | 2 | 179 |
| 17 | F–1 | 2 | 220 |
| 18 | F#–1 | 3 | 8 |
| 19 | G–1 | 3 | 54 |
| 20 | G#–1 | 3 | 103 |
| 21 | A–1 | 3 | 155 |
| 22 | A#–1 | 3 | 210 |
| 23 | B–1 | 4 | 12 |
| 24 | C–2 | 4 | 73 |
| 25 | C#–2 | 4 | 139 |
| 26 | D–2 | 4 | 208 |
| 27 | D#–2 | 5 | 25 |
| 28 | E–2 | 5 | 103 |
| 29 | F–2 | 5 | 185 |
| 30 | F#–2 | 6 | 16 |
| 31 | G–2 | 6 | 108 |
| 32 | G#–2 | 6 | 206 |
| 33 | A–2 | 7 | 53 |
| 34 | A#–2 | 7 | 163 |
| 35 | B–2 | 8 | 23 |
| 36 | C–3 | 8 | 147 |
| 37 | C#–3 | 9 | 21 |
| 38 | D–3 | 9 | 159 |
| 39 | D#–3 | 10 | 60 |
| 40 | E–3 | 10 | 205 |
| 41 | F–3 | 11 | 114 |
| 42 | F#–3 | 12 | 32 |
| 43 | G–3 | 12 | 216 |

| Note | Note–Octave | Hi Freq | Low Freq |
|---|---|---|---|
| 44 | G#–3 | 13 | 156 |
| 45 | A–3 | 14 | 107 |
| 46 | A#–3 | 15 | 70 |
| 47 | 8–3 | 16 | 47 |
| 48 | C–4 | 17 | 37 |
| 49 | C#–4 | 18 | 42 |
| 50 | D–4 | 19 | 63 |
| 51 | D#–4 | 20 | 100 |
| 52 | E–4 | 21 | 154 |
| 53 | F–4 | 22 | 227 |
| 54 | F#–4 | 24 | 63 |
| 55 | G–4 | 25 | 177 |
| 56 | G#–4 | 27 | 56 |
| 57 | A–4 | 28 | 214 |
| 58 | A#–4 | 30 | 141 |
| 59 | 8–4 | 32 | 94 |
| 60 | C–5 | 34 | 75 |
| 61 | C#–5 | 36 | 85 |
| 62 | D–5 | 38 | 126 |
| 63 | D#–5 | 40 | 200 |
| 64 | E–5 | 43 | 52 |
| 65 | F–5 | 45 | 198 |
| 66 | F#–5 | 48 | 127 |
| 67 | G–5 | 51 | 97 |
| 68 | G#–5 | 54 | 111 |
| 69 | A–5 | 57 | 172 |
| 70 | A#–5 | 61 | 126 |
| 71 | 8–5 | 64 | 188 |
| 72 | C–6 | 68 | 149 |
| 73 | C#–6 | 72 | 169 |
| 74 | D–6 | 76 | 252 |
| 75 | D#–6 | 81 | 161 |
| 76 | E–6 | 86 | 105 |
| 77 | F6 | 91 | 140 |
| 78 | F#–6 | 96 | 254 |
| 79 | G–6 | 102 | 194 |
| 80 | G#–6 | 108 | 223 |
| 81 | A–6 | 115 | 88 |
| 82 | A#–6 | 122 | 52 |
| 83 | 8–6 | 129 | 120 |
| 84 | C–7 | 137 | 43 |
| 85 | C#–7 | 145 | 83 |
| 86 | D–7 | 153 | 247 |
| 87 | D#–7 | 163 | 31 |
| 88 | E–7 | 172 | 210 |
| 89 | F–7 | 183 | 25 |
| 90 | F#–7 | 193 | 252 |
| 91 | G–7 | 205 | 133 |
| 92 | G#–7 | 217 | 189 |
| 93 | A–7 | 230 | 176 |
| 94 | A#–7 | 244 | 103 |

## Envelope rates

| Value Dec | Value Hex | Attack Rate (Time/Cycle) ms | Decay/Release Rate (Time/Cycle) ms |
|---|---|---|---|
| 0 | 0 | 2 | 6 |
| 1 | 1 | 8 | 24 |
| 2 | 2 | 16 | 48 |
| 3 | 3 | 24 | 72 |
| 4 | 4 | 38 | 114 |
| 5 | 5 | 56 | 168 |
| 6 | 6 | 68 | 204 |
| 7 | 7 | 80 | 240 |
| 8 | 8 | 100 | 300 |
| 9 | 9 | 250 | 750 |
| 10 | A | 500 | 1.5 sec |
| 11 | B | 800 | 2.4 sec |
| 12 | C | 1 sec | 3 sec |
| 13 | D | 3 sec | 9 sec |
| 14 | E | 5 sec | 15 sec |
| 15 | F | 8 sec | 24 sec |

## ADSR settings

We'll explain what ADSR actually stands for in the next chapter. For now, it suffices to show you what values control what effects.

**ATTACK/DECAY RATE SETTINGS**

| ATTACK/DECAY | SETTING | HIGH ATTACK | MEDIUM ATTACK | LOW ATTACK | LOWEST ATTACK | HIGH DECAY | MED. DECAY | LOW DECAY | LOWEST DECAY |
|---|---|---|---|---|---|---|---|---|---|
| VOICE 1 | 54277 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| VOICE 2 | 54284 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| VOICE 3 | 54291 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

**SUSTAIN/RELEASE RATE SETTINGS**

| SUSTAIN/ CONTROL | RELEASE SETTING | HIGH SUSTAIN | MEDIUM SUSTAIN | LOW SUSTAIN | LOWEST SUSTAIN | HIGH RELEASE | MED. RELEASE | LOW RELEASE | LOWEST RELEASE |
|---|---|---|---|---|---|---|---|---|---|
| VOICE 1 | 54278 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| VOICE 2 | 54285 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| VOICE 3 | 54292 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

# SID chip description

The SID chip, which allows us to create all our weird and wonderful noises, has 28 registers that it will allow us to alter. In Chapter 4 we'll explain what they all do, but for now:

Register Description
(54272+)

00 Low frequency value of note for voice 1

01 High frequency value of note for voice 1

02 Low pulse rate for voice 1

03 High pulse rate for voice 1

04 Waveform for voice 1

05 Attack/decay for voice 1

06 Sustain/release for voice 1

07 Low frequency value of note for voice 2

08 High frequency value of note for voice 2

09 Low pulse rate for voice 2

10 High pulse rate for voice 2

11 Waveform for voice 2

12 Attack/decay for voice 2

13 Sustain/release for voice 2

14 Low frequency value of note for voice 3

15 High frequency value of note for voice 3

16 Low pulse rate for voice 3

17 High pulse rate for voice 3

18 Waveform for voice 3

19 Attack/decay for voice 3

20 Sustain/release for voice 3

21 High frequency cut-off

22 Low frequency cut-off

23 Turn on filtering

24 Set volume for all three voices, plus select filter type

25 Access to output of envelope generator of voice 3

26 Access to Y potentiometer reading on pin 23

27 Digitised output from voice 3

28 Digitised output from envelope generator number 3

# 4
# What the 64 Lets Us Do

From the tables in the previous chapter you might surmise that the 64 lets us do quite a lot, and you'd be right. But before we can even play a note, there are some more definitions and facts to be learnt.

## The 64: an overview

The Commodore 64 has a remarkably gifted sound capability, courtesy of the 6581 SID chip. You don't need to be a musician to use this machine, although if you are, the keyboard and other musical programs will probably help you begin to understand how it all works, and certainly the table of musical notes given in Chapter 3 will be of great use when transposing music.

What you will need to know are the ways in which the Commodore 64 uses its powerful SID chip, the memory registers that are affected, and the power at your disposal. SID can control three voices, each one having a practical octave range of 8 octaves. Unfortunately we don't have separate volume controls over each voice, but have to change them all at the same time. This is not unusual among musical chips on home micros, but it is unusual on the 64, since this is a custom-built chip. There are also some locations (one for each voice) which don't actually do anything, which leads one to suspect that someone may have wanted to include separate volume controls and thus left the space in there to provide them, but someone else decided not to. A shame, because that is about all this chip lacks.

For each voice we have control over four waveforms, namely triangle, sawtooth, variable pulse, and noise. Our three envelope generators, combined with ring modulation, programmable filters and the rest give SID the same sort of capabilities as many a more expensive dedicated synthesiser. It is a relatively easy matter to produce successful impersonations of many musical instruments, and later on we'll be doing just that, as well as giving you a number of sample tunes and programs to type in.

## Three voices

The waveform for each of the three voices can be changed using the appropriate register, and each voice can independently mimic a wide variety of musical instruments. In order to do that, we have to adjust a variety of settings, and we'll start by looking at attack, decay, sustain and release, collectively known as ADSR.

These measure the length of time it takes a note to come to its maximum volume, and the time taken to go back to total silence, and then the length of time for which it will maintain its maximum volume before letting go again.

In Chapter 3 we gave the values for adjusting these settings, and these values are combined in the following way. If, for voice one, we POKE 54277 with 16, we have the lowest attack rate, and no decay. POKEing it with 20 would give us the same attack rate, but this time a medium decay, as 20 is a combination of the settings for 16 and 4. POKEing 54272 with 72 would give us a medium attack and a high decay, and so on.

Sustain/release works in exactly the same way. POKEing 54278 with 40 would give us a low sustain and a high release, as 40 is a combination of 32 (low sustain) and 8 (high release).

Before even playing a note, we've got to know how to turn the voices on, and a look at the table in the previous chapter will show us that to set the volume (always wise!) we need to POKE 54296, and we can use any number from 0 (silence) through to 15 (maximum volume). This location controls a few other things as well, as we'll be seeing in later chapters.

Then we must select the waveform we require. For voice one, this is achieved by altering location 54276, and there are four values we can put in there:

17 : gives us a triangle waveform.

33 : gives us a sawtooth waveform.

65 : gives us a pulse waveform.

129 : generates white noise.

We'll go into waveforms in more detail later, but all we need to know now is the actual note that we want to play, and this is got by POKEing locations 54273 and 54272 for voice one with the high frequency and the low frequency respectively of the note to be played.

By playing with the ADSR settings and the waveforms you should be able to work out one or two instruments for yourself, and the 64 can manage to produce impersonations of such instruments as banjos, pianos, harpsichords, accordions, and many more.

So far we haven't really mentioned the third waveform produced by POKEing S+4 with 65. This is because this pulse wave requires a couple more parameters than the other three, known as the pulse width.

You can see the memory locations that need to be altered by looking at the chart in Chapter 3, and the values that go into the locations behind on the sort of pulse wave you want to create: more in a minute.

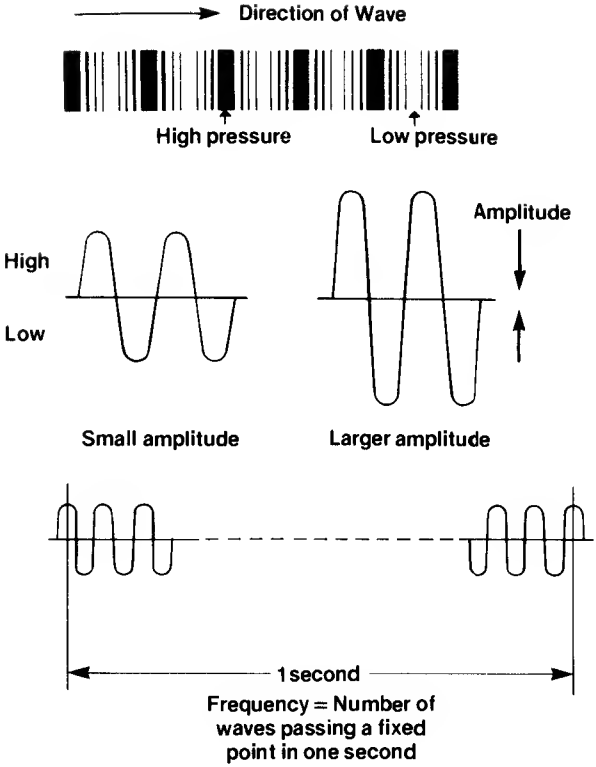
## Down to business

To use music, or noise, properly on the 64 you need to know a little about the theory behind it all. Which is not to say that, if you don't know the first thing about sound waves, or what the difference between a sawtooth and a triangle is, or you think envelope generating is something to do with a trip to the post office, you won't get a lot of enjoyment out of the 64's SID chip. Using just what we've learnt in the first section of this chapter should keep you busy impersonating various instruments, generating wonderful explosions, or whatever, for a long time to come. However, the SID chip is capable of a lot more than the little bit we've looked at so far.

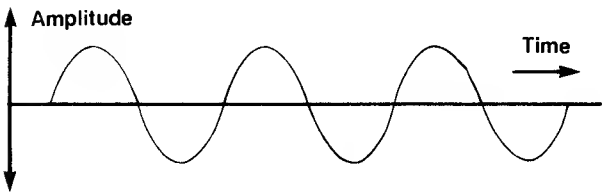
### Different waveforms

As we've seen, frequency is a measure of the number of waves passing a fixed point in one second. There are other features associated with every form of wave, and the principal of these is amplitude: simply a measure of how large or small the wave is.

Thus, waves produced by dropping a stone into a pond will have a small amplitude, whereas waves down on the local beach will generally have a large amplitude. The following diagram should help to illustrate this:




Another term you might come across is pitch. This is just another way of reckoning up the frequency of a note: notes with a high frequency are said to have a high pitch, and those with a low frequency to have a low pitch. However, notes of the same pitch played by different instruments can sound remarkably different. This is due to the shape of the waveform that the instrument is generating, and all waveforms are made up of a variety of different sine waves.

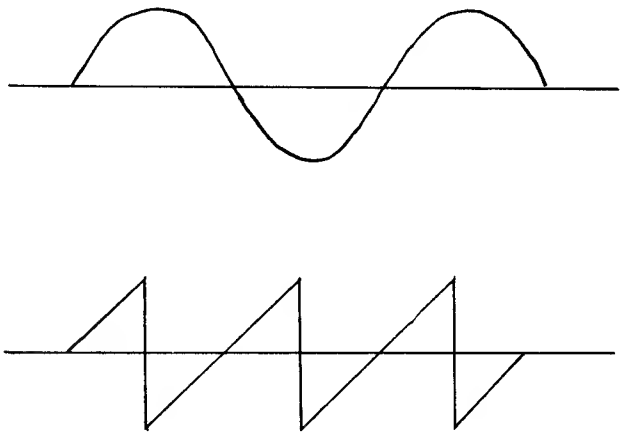


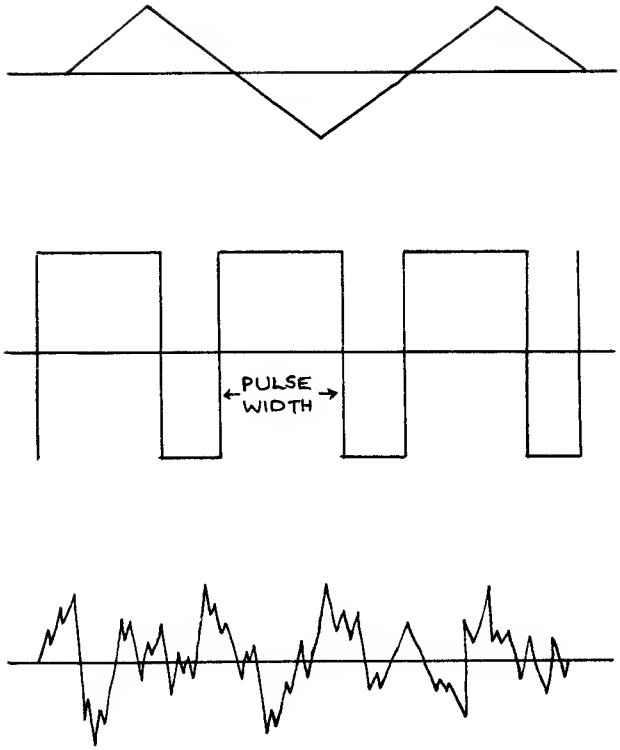
This is the waveform on which all others are based, and any other waveform can always be reduced to a mixture of sine waves of different frequencies. The main wave is called the fundamental one, and determines the pitch of the note, or frequency; this is often called the fundamental frequency.

The rest of the waveform consists of variations on this fundamental frequency, and these variations are called harmonics. Thus a note consists of its fundamental frequency, and all the harmonics of that frequency. The frequencies of the harmonics are calculated from the fundamental: that of the second harmonic is twice that of the fundamental (or first harmonic), the third harmonic has three times the frequency, and so on.

The number and mixture of harmonics go together to make up the overall quality, or timbre, of the note. The amount of each harmonic present is related to the square of the harmonic number, so that the second harmonic is 1/(2*2), or one quarter, as loud as the first one, the third is 1/(3*3), or one ninth as loud as the first one, and so on.

Now that we know all that, let's have a look at the shape of the waveforms generated by the Commodore 64.






The first one is a sine wave, reproduced here for reference. The second is the sawtooth wave, and a look at the diagram will readily show you how it gets its name. Sawtooths contain all the harmonics, and can be tremendously difficult to calculate, which is why precise musical impersonation can be rather difficult. The third diagram shows the triangular wave. These are much easier to calculate, as they contain only the odd harmonics (i.e. the first, third, fifth, and so on). The fourth diagram is that of the pulse waveform, and when using this on the 64 we have to know a little bit more about it.

On the diagram we've indicated the pulse width, or the width of each pulse of the waveform. These must be calculated for use on the 64, and the formula goes as follows:

P = PW/40.95 %

where PW is the actual pulse width. If this is equal to 2048, for instance, we have a pure square wave.

The pulse width is made up of two components as far as the 64 is concerned, a high and a low width, and these values for voice 1 are POKEd into locations 54274 and 54275 for voice 1. The high width lies between 0 and 15, and the low width between 0 and 255.

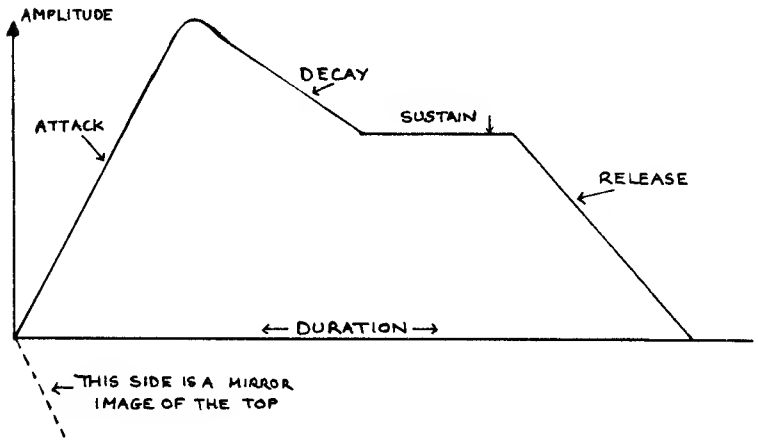
Looking at our earlier square wave, to program this into the machine we'd have to:

```
POKE 54275,8:POKE 54274,0
```

in the traditional high-low form, as $0800 is equal to 2048 in decimal terms.

The final waveform is the noise one, a purely random collection of waves, which is used mainly for creating sound effects and frightening the neighbours.

To look at each individual note, as we've seen, we need to study the attack/decay and sustain/release of the note. In graphical form, a typical note might look something like this:

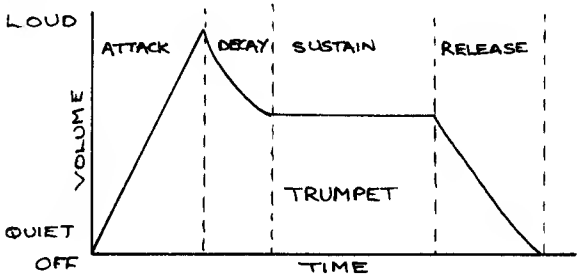


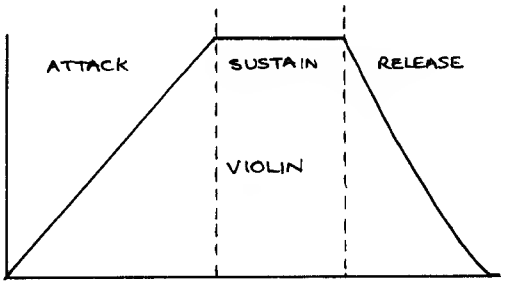

We've plotted amplitude against time, and shown the relevant four sections of the note.

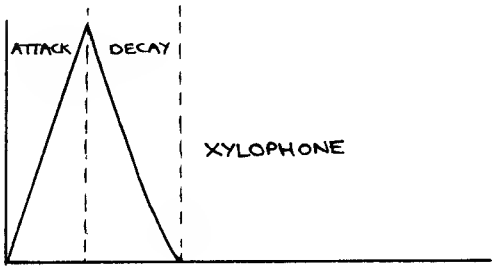
Knowing all this, perhaps you might like to turn to some of our Close Encounter programs and amend them to make them sound rather more interesting!

### Envelope generation

We have in fact already covered this, so it can't be as frightening as it sounds. Envelope generation is just the term used to describe making up the attack/decay and sustain/release parts of a note.



Just by altering these four parameters, as we've already seen, we can produce some exciting effects.

### Filtering

There are three different types of filter on the 64 (see earlier for the values of the relevant memory locations), and these three filters affect the shape of the harmonic content of a waveform.

The three filters are a high pass one, which passes all the frequencies above a POKEd cut-off point, a low-pass filter, which, as you might guess, passes frequencies below a specified cut-off point, and a band-pass filter, which passes everything with a specified narrow band, and attenuates everything else.

These can be combined in a variety of ways, and we'll be taking a closer look at this in Chapter 9.

## Conclusion

That rounds off our overview of the SID chip, and should give you some idea of the power behind it. But we haven't even made a sound yet, I hear you cry. Don't despair, Chapter 5 will set that to rights as we start, at last, to produce some musical notes.

# 5
# In Which We Play a Tune

So far we've had nothing but theory, so it's about time that we began putting that theory into practice and making a little bit of noise. The remaining chapters in this book contain a wealth of musical and not-so-musical programs, but for now we're going to content ourselves with playing a note, and then, sticking with just one voice, producing a tune or two.

To produce even a single note we must put values into a lot of memory locations, and we have to be careful about the order in which we alter those locations as well. Thankfully, for playing a complete tune, we usually only have to alter two locations continually, so after the hard work of setting everything up has been done, we don't need to worry about the rest of them again.

First of all, make sure that the volume on your television set is turned up to an acceptable level. An obvious point maybe, but I have seen people spend a good five or ten minutes tinkering with a program trying to work out why it isn't issuing a sound, only for some amused onlooker to walk up and turn the volume control on the set.

Secondly, it makes sense to define a base value that we'll refer to, rather than keep typing out extremely large numbers that are very easy to get muddled up. So, for want of a better letter, we'll use the letter V to be the base location for voice one, and we'll define V in the following way:

```
10 V=54272
11 REM SET BASE VALUE FOR VOICE ONE
```

Having got that out of the way, we'll need to worry about the attack, decay, sustain and release settings. A long decay can mean that a note takes an infinity to die away, and if it's a high-pitched note every dog in the vicinity will be driven slowly round the bend. So, incidentally, will you.

We'll go for a few middle-of-the-road settings:

```
20 POKEV+5,9:POKEV+6,0
21 REM SET A LOW ATTACK, REASONABLE DECAY, ZERO
SUSTAIN AND RELEASE
```

What sort of waveform should we have? We'll go for a pulse one, and get all the settings in the one program. Choosing a pulse waveform means that we'll have to choose a pulse width as well. The following lines will give us a sound something like a piano:

```
30 POKEV+4,0:POKEV+2,255:POKEV+3,0:POKEV+4,65
31 REM MAXIMIMUM LOW ORDER, ZERO HIGH ORDER, AND
SELECT PULSE WAVEFORM
```

Now all we need to worry about is turning on the volume:

```
40 POKEV+24,15
41 REM SET VOLUME AT MAXIMUM
```

Now we can play a note. Let's go for middle C: and a glance at the appropriate table in Chapter 3 shows us that middle C has a value of 4 for the high order frequency, and 73 for the low order frequency. So:

```
50 POKEV+0,73:POKEV+1,4
51 REM MIDDLE C
```

There we have it: a Commodore 64 playing middle C and sounding very like a piano. Well, similar anyway.

By altering the pulse width you can produce some quite interesting results. Insert the following two lines into your program, and change line 30 as indicated:

```
25 FORI=1TO255
55 NEXTI

30 POKEV+4,0:POKEV+2,255:POKEV+3,I:POKEV+4,65
```

This sounds weird and wonderful, as we rapidly alter the pulse width. For a more gradual change, alter lines 25, 30 and 55 as indicated.

```
25 FORI=1TO255:FORJ=1TO255
30 POKEV+4,0:POKEV+2,J:POKEV+3,I:POKEV+4,65
55 NEXTJ,I
```

This alters both the high pulse width and the low pulse width, and sounds quite interesting. Altering the pulse width in this way is just one of the methods used to produce some amusing sounds. Using different waveforms in this program will sound pretty boring, since they don't have any pulse widths to alter, and so all that will happen will be a collection of strange noises. However, if one were to change the program slightly, and start altering the values POKEd in for the high and low frequencies of the note itself— well, just type it in and listen to what happens.

```
10 V=54272
20 POKEV+5,9:POKEV+6,0
30 FORI=1TO255:FORJ=1TO50
40 POKEV+4,0:POKEV+4,17
50 POKEV+24,15
60 POKEV+1,J:POKEV+0,I
70 NEXTJ,I
```

This will eventually begin to sound pretty unbearable as I approaches values over about 80 or so. A slightly more interesting approach would be as follows:

```
10 V=54272
20 POKEV+5,36:POKEV+6,36
25 POKEV+24,15
30 FORI=0TO50:FORJ=0TO50
40 POKEV+4,0:POKEV+4,33
50 POKEV+1,J:POKEV+0,I
60 NEXTJ
70 FORJ=50TO0STEP-1
80 POKEV+4,0:POKEV+4,33
90 POKEV+1,J:POKEV+0,I
100 NEXTJ
110 NEXTI
120 GOTO30
```

This results in a never-ending supply of noise, and illustrates how progressive loops can be used to produce some, at times, bizarre results. If we were to alter the pulse width (using a pulse waveform, of course) in conjunction with these loops, the sounds would get very interesting.

## Noise

Using the noise waveform, we can go even further:

```
10 V=54272
20 POKEV+5,36:POKEV+6,36
25 POKEV+24,15
30 FORI=0TO50:FORJ=0TO50
40 POKEV+4,0:POKEV+4,129
50 POKEV+1,J:POKEV+0,I
60 NEXTJ
70 FORJ=50TO0STEP-1
80 POKEV+4,0:POKEV+4,129
90 POKEV+1,J:POKEV+0,I
100 NEXTJ
110 NEXTI
120 GOTO30
```

So you see, just a couple of simple changes to the listing, and a whole new world opens up in terms of bizarre noises. Without knowing anything about ring modulation and synchronisation (to say nothing of filtering techniques) we can already produce some very interesting sounds. What we haven't produced yet, though, is a tune, and one easy way of doing this is as follows.

## In tune

We could have a whole series of POKE statements for each and every note, but that could get a little tedious after a while, and so it makes rather more sense to have all the notes stored as data statements.

Using the table in Chapter 3, we could easily produce a program that would play the scale of C major.

```
10 V=54272
20 POKEV+5,9:POKEV+6,0
30 POKEV+24,15
40 POKEV+2,255:POKEV+3,0:POKEV+4,65
50 READA,B,C
60 IFA=-1THEN100
70 POKEV+4,0:POKEV+4,65
80 POKEV+1,B:POKEV+0,C:FORI=1TOA*50:NEXT
```

```
90 GOTO50
100 FORI=0TO24:POKEV+I,0:NEXT
1000 DATA5,4,73,5,4,208,5,5,103,5,5,185
1010 DATA5,6,108,5,7,53,5,8,23,5,8,147
1020 DATA-1,0,0
```

Or, if you want something a bit more tuneful, how about this:

```
10 V=54272
20 POKEV+5,9:POKEV+6,0
30 POKEV+24,15
40 POKEV+2,255:POKEV+3,0:POKEV+4,65
50 READA,B,C
60 IFA=-1THEN100
70 POKEV+4,0:POKEV+4,65
80 POKEV+1,B:POKEV+0,C:FORI=1TOA*50:NEXT
90 GOTO50
100 FORI=0TO24:POKEV+I,0:NEXT
1000 DATA5,4,73,5,5,5,103,5,6,108,5,7,53
1010 DATA5,7,163,5,7,53,5,6,108,5,5,103
1020 DATA-1,0,0
```

The start of a simple boogie riff. This method of playing tunes will be returned to in the next chapter when we start considering the use of more than one voice.

## Further techniques

As we are only using the one voice here, it is a little bit difficult to consider any of the more advanced features of the machine, since most of them rely on'bouncing' one voice off another. Synchronisation, for instance, literally synchronises one voice with another, and if one voice is silent (as so far we're only using one voice in total) not a lot is going to happen.

The most startling effects that can be achieved with one voice (be it voice one, two or three), depend on the use of pulse or noise waveforms. If we're using noise, then we can produce effects rather like explosions, percussion instruments, or, with a little bit of fiddling around, you can produce a passable impersonation of a game of squash being played!

With pulse waveforms, our effects come chiefly from altering the pulse width and studying the changes in the note as the width changes.

Rapid effects come from altering the high frequency of the note, more subtle ones from altering the low frequency. Filtering, and altering the frequency at which a note is cut off, can also be quite interesting, but we'll consider this in more detail in Chapter 9.

## Conclusion

There is a lot that can be done with just one voice, but two or more are always better. One voice is chiefly used to play a background tune in a game, say, as we'll be seeing later on. Indeed, you could have two voices bouncing off each other, and reserve the third for the special 'explosion' noises, or whatever.

But with one voice, it's worth playing about with pulse widths and noise waveforms just to see (or rather, hear) what happens.

# 6
# Three-Part Harmony

## Introduction

From here on we're going to start using some slightly longer programs to illustrate the techniques that we'll be talking about. If at any time your nerve fails you, and the thought of typing in yet another line of code is beginning to drive you to despair, remember that all the longer programs are available on one cassette from the publishers, Duckworth, at a price of £7.95.

In the previous chapter we were using just one voice to play notes or a simple tune, but with three voices available to us on the 64 it obviously makes sense to strive for greater things. On a fairly minor level that's what we'll be looking at in this chapter, since we're going to be concentrating on getting the groundwork out of the way before settling down to the more complicated side of things.

Using the techniques explored in the last chapter, we can use all three voices on the 64 to play a familiar little tune: an ordinary boogie riff. I keep picking this as an example because it's a straightforward tune, because anyone can tell if they've typed in a wrong note in a tune as simple as this, and finally because I'm in a similar position to Jimmy Hill when he talks about football: fine on theory, not so good on practice. My ear for music is, well, not the best one you're ever likely to come across!

## Sing a song

Although the main program here is fairly short, you might have a bit of trouble with all the data statements, since there are quite a lot of them. Still, perseverance has its own reward as they say, so stick with it.

As with most of the programs presented in this book, a line-by-line explanation follows the listing.

```
90 POKE53280,0:POKE53281,0:PRINT"[CLR,YEL]":
100 PRINT"MORE MUSIC FROM MR. BUTTERFIELD
110 I1=54272:I2=54279:I3=54286
120 H1=I1+1:H2=I1+1:H3=I3+1
130 V1=I1+4:V2=I2+4:V3=I3+4
140 POKE I1+24,15
150 POKEV1+1,9:POKEV1+2,0
160 POKEV2+1,36:POKEV2+2,36
170 POKEV3+1,18:POKEV3+2,250
175 POKEI1+2,255:POKEI1+3,0
180 T=TI
200 POKEV1,64:POKEV2,128:POKEV3,32
210 READS:IFS=0THENRESTORE:GOTO210
220 READ X1,Y1,X2,Y2,X3,Y3
230 IFX1THENPOKEH1,X1:POKEI1,Y1:POKEV1,65
240 IFX2THENPOKEH2,X2:POKEI2,Y2:POKEV2,129
250 IFX3THENPOKEH3,X3:POKEI3,Y3:POKEV3,33
260 T=T+S
270 IFT>TITHEN270
280 GOTO200
290 FORJ=I1TO54296:POKEJ,0:NEXT
300 DATA 15,17,37,68,149,4,73
310 DATA 15,21,154,0,0,0,0
320 DATA 15,25,177,0,0,6,108
330 DATA 15,28,214,0,0,0,0
340 DATA 15,30,141,115,88,7,163
350 DATA 15,28,214,0,0,0,0
360 DATA 15,25,177,0,0,6,108
370 DATA 15,21,154,0,0,0,0
380 DATA 15,17,37,68,149,4,73
390 DATA 15,21,154,0,0,0,0
400 DATA 15,25,177,0,0,6,108
410 DATA 15,28,214,0,0,0,0
420 DATA 15,30,141,115,68,7,163
430 DATA 15,28,214,0,0,0,0
440 DATA 15,25,177,0,0,6,108
450 DATA 15,21,154,0,0,0,0
460 DATA 15,22,227,91,140,5,185
470 DATA 15,28,214,0,0,0,0
480 DATA 15,34,75,0,0,8,147
490 DATA 15,38,126,0,0,0,0
500 DATA 15,40,200,163,31,10,60
510 DATA 15,38,126,0,0,0,0
520 DATA 15,34,75,0,0,8,147
530 DATA 15,28,214,0,0,0,0
540 DATA 15,17,37,68,149,4,73
550 DATA 15,21,154,0,0,0,0
560 DATA 15,25,177,0,0,6,108
570 DATA 15,28,214,0,0,0,0
580 DATA 15,30,141,122,52,7,163
590 DATA 15,28,214,0,0,0,0
600 DATA 15,25,177,0,0,6,108
```

```
610 DATA 15.21,154.0,0,0,0
620 DATA 15.25.177,102,194,6.108
630 DATA 15.32,94,0,0,0,0
640 DATA 15.38,126,0,0,9,159
650 DATA 15.32,94,0,0,0,0
660 DATA 15,22,227,91,140,5,185
670 DATA 15.28,214,0,0,0,0
680 DATA 15,34,75,0.0,8,147
690 DATA 15.28.214,0,0,0,0
999 DATA 0,0,0,0,0,0,0
```

**Explanation**

Line 90 : Change the border and background colours to black, clear the screen, and turn the print colour to yellow.

Line 100 : So you know who to blame for the program. Blame me for the tune.

Line 110 : Declare three variables, for voice one, two and three respectively.

Line 120 : Declare three more variables. These form the high order value of the note for voices one, two and three respectively.

Line 130 : And three more variables, which are the locations to be altered for the waveform for voice one, two and three respectively.

Line 140 : Set the volume to maximum.

Line 150 : Set ADSR cycle for voice one. Here we have a fast attack, a slight decay, and no sustain or release.

Line 160 : Ditto for voice two, giving us middle-of-the-road values for all four parameters in the cycle.

Line 170 : And again for voice three, with a fast attack, a medium decay, and a medium sustain and release as well.

Line 175 : Since voice one is going to use the pulse waveform, we need to set the pulse width as well, which is the function of this line.

Line 180 : Set a variable T to equal the current time TI.

Line 200 : Switch all three waveforms off before playing a note. This

should always be done on the 64, as should setting the ADSR parameters before choosing the waveform. If you don't adhere to this, not very much will happen.

Line 210 : Read the first item of data, which determines how long the notes will be held for before being switched off. If the data item is zero, then RESTORE everything and go back to the start again. If you altered this line to read:

```
210 READS:IFS=0THEN290
```

this would ensure that the song cycle would be played once only, after which everything would be switched off by line 290.

Line 220 : Read the high order and low order note values for voices one, two and three respectively.

Line 230 : If X1, the high order, for voice one has a value, then POKE it in there, along with the low order value, followed by setting the waveform to be on. If it doesn't have a value, i.e. the data item was zero, fall through to the next line.

Line 240 : Ditto for voice two.

Line 250 : And again for voice three.

Line 260 : Add the delay time S to the variable T set earlier.

Line 270 : If this new value is greater than the current time TI then hang around until it isn't.

Line 280 : Trot back to line 200 for the next item of data.

Line 290 : Optional loop to switch everything off. Could be called up from line 210 if required, and it's also useful to have it there in case you decide to break into the program, in which case a GOTO290 will set everything to zero. It is always a good idea, in the case of sound, to turn everything off at the end of a program.

Lines 300 onwards : Data for the tune.

## Other methods

That is, of course, just one way of doing things. In Chapter 10 we'll

see a more advanced method, in which voices three and two are used to provide a background rhythm, and only voice one is directly controlled and altered.

Using data can, however, lead to problems, especially if there is a lot of other data in the program as well. If you're having a program with sprites and/or user-defined characters, it is easily possible for the wrong data to be read at the wrong time, and the resulting cacophony would have you reaching for the volume control before you could say Paul McCartney. If the 64 had a programmable RESTORE command like many other machines (i.e. you could have a line that restored the data pointer to a specific line of data) then we'd probably be okay, but RESTORE on the 64 only takes us back to the very start of the data, which is not always a good idea.

If you've got plenty of memory left, as is the case in the program at the end of this chapter, you can resort to using strings instead of data. By defining a string to be equal to, say, "5,4,73,5,4,73" etc., a simple bit of coding could remove the commas from that string, split it up into three numbers at a time, and then use code similar to that in the earlier program to play the notes.

And how, precisely, would we go about doing this? Well, let's take a look at the program and see.

## Musical Hangman

This is basically a variation on a theme, since it is nothing more special than the old game of Hangman. However, it does use some interesting programming techniques, and most of the words in it are musical ones, and so it is worthy of its place here.

```
5 POKE53281,0:POKE53280,0:POKE53272,23:PRINT"[CLR,
YEL]MUSICAL HANGMAN.
15 PRINT"[2CD]THIS IS BASICALLY THE OLD GAME OF
    HANGMAN, WITH ONE DIFFERENCE.
20 PRINT"[CD]ALL THE WORDS YOU'LL BE ASKED TO
   IDENTIFY ARE EITHER MUSICAL ";
25 PRINT"TERMS,       INSTRUMENTS, OR NAMES OF WELL
-KNOWN      MUSICIANS OR GROUPS."
30 PRINT"[CD]YOU'LL HAVE NINE CHANCES TO GUESS EAC
H  WORD AS IT APPEARS.  JUST ";
35 PRINT"ENTER YOUR     LETTER GUESS BY PRESSING TH
E APPROPRIATEKEY ON THE ";
```

```
40 PRINT"KEYBOARD, AND YOUR PROGRESS  (OR LACK OF
IT) WILL BE MARKED";
45 PRINT"           ACCORDINGLY."
50 T=8:GOSUB62000:GOSUB63000
55 PRINT"[2CD]PRESS 'SPACE' WHEN READY TO BEGIN."
60 GETSP$:IFSP$<>" "THEN60
65 PRINT"[2CD]OKAY, JUST READING THE DATA IN."
70 DIMW$(171),G$(26):FORI=1TO171:READW$(I):NEXT
74 FORI=1TO26:G$(I)="":NEXT:H$=""
75 PRINT"[CLR,RVS]MUSICAL HANGMAN.":POKES+24,15
80 J=INT(RND(.5)*171+1):W$=W$(J):G=1
82 A=ASC(MID$(W$,1,1))-128:A$=CHR$(A):W$=A$+MID$(W
$,2)
84 FORI=1TOLEN(W$):H$=H$+"*":NEXT
85 PRINT"[CD]YOUR WORD HAS";LEN(W$);"LETTERS IN IT
."
90 PRINT"[2CD]LETTERS USED:"
95 FORI=1TO9:IFG$(I)<>""THENPRINTG$(I);
100 NEXT:PRINT
105 FORI=10TO18:IFG$(I)<>""THENPRINTG$(I);
110 NEXT:PRINT
115 FORI=19TO26:IFG$(I)<>""THENPRINTG$(I);
120 NEXT:PRINT
122 IFH$=W$THENPRINT"[CLR,2CD]YOU'VE GOT IT!":PRIN
T"[CD]THE WORD WAS ";W$;"!":GOTO2000
125 PRINT"[2CD]ENTER YOUR GUESS:"
130 PRINT:PRINT:PRINTH$
135 FORI=1TO10:GETA$:NEXT
136 POKES+4,0
140 GETA$:IFA$=""THEN140
142 POKES+1,2+G:POKES,2
145 A=ASC(A$):IFA<65ORA>90THEN140
146 POKES+4,65
150 G$(G)=CHR$(A):G=G+1
155 FORI=1TOLEN(W$)
160 IFMID$(W$,I,1)=G$(G-1)THEN500
165 NEXT
170 IFZ=0THENGOSUB600:PRINT"[HOME]":GOTO85
175 Z=0:PRINT"[HOME]":GOTO85
500 H$=MID$(H$,1,I-1)+G$(G-1)+MID$(H$,I+1)
502 Z=Z+1
510 GOTO165
600 ZZ=ZZ+1
604 POKEV+21,255:POKEV+23,225:POKEV+29,225
606 ONZZGOTO610,620,630,640,650,660,670,680,690
610 POKEV+0,240:POKEV+1,180:RETURN
620 POKEV+14,240:POKEV+15,138:RETURN
630 POKEV+12,240:POKEV+13,138:RETURN
640 POKEV+10,240:POKEV+11,138:RETURN
650 POKEV+6,234:POKEV+7,150:RETURN
660 POKEV+4,234:POKEV+5,150:RETURN
670 POKEV+2,234:POKEV+3,150:RETURN
```

```
680 POKEV+8,234:POKEV+9,150:RETURN
690 PRINT"[CLR,2CD]OH DEAR!!":PRINT"[CD]THE WORD W
AS ";W$;"!"
695 GOSUB1000:POKEV+3,0:POKEV+5,0:POKEV+7,0:POKEV+
8,240
700 POKES+4,0:POKES+4,65:FORI=150TO200:POKEV+9,I:P
OKES+1,200-I:POKES,200-I:NEXT
704 FORI=1TO1500:NEXT
705 PRINT"[CLR,2CD]ANOTHER GO (Y OR N)?":ZZ=0:G=1
710 FORI=1TO10:GETN$:NEXT
720 GETN$:IFN$="N"THENFORI=0TO15:POKEV+I,0:NEXT:PO
KEV+21,0:POKEV+28,0:END
730 IFN$<>"Y"THEN720
735 FORI=0TO15:POKEV+I,0:NEXT
740 GOTO74
1000 REM DEATH TUNE
1002 Z1$="5,4,208,5,4,208,1,4,208,6,4,208,5,5,180,
1,5,103,5,5,103,"
1004 Z1$=Z1$+"1,4,208,4,4,208,3,4,73,15,4,208,0,0,
0"
1006 GOSUB11000:RETURN
2000 Z1$=",5,4,73,,5,4,73,,5,5,103,,5,6,108,,5,8,1
4,,5,7,53,5,6,108,,5,4,73,"
2001 Z1$=Z1$+",5,5,103,,5,6,108,,5,10,205,5,9,159,
,5,8,147,,5,8,23,,5,8,147,"
2002 Z1$=Z1$+",5,7,53,,5,8,23,,5,6,108,,5,7,53,,5,
5,185,,5,6,108,,5,5,103,"
2003 Z1$=Z1$+",5,5,185,,5,4,208,,5,5,103,5,4,73,0"
2004 GOSUB11000:POKEV+21,0:GOTO704
11000 A$="":B$="":A=0:B=0:C=0:D=0:E=0:REMMUSIC
11004 FORJ=1TOLEN(Z1$)
11005 B$=MID$(Z1$,J,1)
11006 IFB$<>","THENA$=A$+B$:GOTO11008
11007 A=VAL(A$):GOSUB11020
11008 NEXT
11010 IFA=0THENC=0:A$="":POKES+4,0:RETURN
11020 C=C+1
11022 IFC=1THEND=A:A$="":RETURN
11024 IFC=2THENB=A:A$="":RETURN
11030 E=A
11032 POKES+1,B:POKES,E:POKES+4,65:FORK=1TOD*50:NE
XTK:C=0:A$="":POKES+4,64:RETURN
29999 END
30000 DATA1, 2 , 7 , 8
30001 DATA0,0,21,0,0,21,0
30002 DATA0,21,0,0,21,0,0
30003 DATA21,0,0,21,0,0,21
30004 DATA0,0,21,0,0,21,170
30005 DATA170,170,170,170,170,170,170
30006 DATA170,170,170,170,170,170,170
30007 DATA170,170,170,170,170,170,170
30008 DATA170,170,170,170,170,170,170
```

```
30009 DATA170,170,170,170,170,170,170,0
30010 DATA1, 2 , 7 , 8
30011 DATA0,40,0,0,40,0,0
30012 DATA40,0,0,20,0,0,215
30013 DATA0,35,85,200,35,85,200
30014 DATA15,85,240,0,85,0,0
30015 DATA85,0,0,85,0,0,170
30016 DATA0,0,85,0,0,85,0
30017 DATA0,65,0,0,65,0,0
30018 DATA0,0,0,0,0,0,0
30019 DATA0,0,0,0,0,0,0,0
30020 DATA1, 2 , 7 , 8
30021 DATA0,40,0,0,40,0,0
30022 DATA40,0,0,20,0,0,215
30023 DATA0,35,85,200,35,85,200
30024 DATA15,85,240,0,85,0,0
30025 DATA85,0,0,85,0,0,0
30026 DATA0,0,0,0,0,0,0
30027 DATA0,0,0,0,0,0,0
30028 DATA0,0,0,0,0,0,0
30029 DATA0,0,0,0,0,0,0,0
30030 DATA1, 2 , 7 , 8
30031 DATA0,40,0,0,40,0,0
30032 DATA40,0,0,20,0,0,215
30033 DATA0,0,85,0,0,85,0
30034 DATA0,85,0,0,85,0,0
30035 DATA85,0,0,0,0,0,0
30036 DATA0,0,0,0,0,0,0
30037 DATA0,0,0,0,0,0,0
30038 DATA0,0,0,0,0,0,0
30039 DATA0,0,0,0,0,0,0,0
30040 DATA1, 2 , 7 , 8
30041 DATA0,40,0,0,40,0,0
30042 DATA40,0,0,20,0,0,215
30043 DATA0,35,85,200,35,85,200
30044 DATA15,85,240,0,85,0,0
30045 DATA85,0,0,85,0,0,170
30046 DATA0,0,85,0,0,85,0
30047 DATA0,65,0,0,65,0,0
30048 DATA195,0,0,195,0,0,195
30049 DATA0,0,130,0,2,130,128,0
30050 DATA1, 2 , 7 , 8
30051 DATA85,85,85,85,85,85,85
30052 DATA85,85,16,20,21,16,20
30053 DATA21,16,5,21,0,5,21
30054 DATA0,1,85,0,1,85,0
30055 DATA0,21,0,0,21,0,0
30056 DATA21,0,0,21,0,0,21
30057 DATA0,0,21,0,0,21,0
30058 DATA0,21,0,0,21,0,0
30059 DATA21,0,0,21,0,0,21,0
30060 DATA1, 2 , 7 , 8
```

```
30061 DATA85,85,85,85,85,85,85
30062 DATA85,85,0,0,21,0,0
30063 DATA21,0,0,21,0,0,21
30064 DATA0,0,21,0,0,21,0
30065 DATA0,21,0,0,21,0,0
30066 DATA21,0,0,21,0,0,21
30067 DATA0,0,21,0,0,21,0
30068 DATA0,21,0,0,21,0,0
30069 DATA21,0,0,21,0,0,21,0
30070 DATA1, 2 , 7 , 8
30071 DATA0,0,21,0,0,21,0
30072 DATA0,21,0,0,21,0,0
30073 DATA21,0,0,21,0,0,21
30074 DATA0,0,21,0,0,21,0
30075 DATA0,21,0,0,21,0,0
30076 DATA21,0,0,21,0,0,21
30077 DATA0,0,21,0,0,21,0
30078 DATA0,21,0,0,21,0,0
30079 DATA21,0,0,21,0,0,21,0
40000 DATA"BREVE","MINIM","CROTCHET","QUAVER","NOT
E","SOUND","STAVE","BAR"
40001 DATA"KEYBOARD","ACCELERANDO","ANIMATO","ARPE
GGIO","TEMPO","PSALM","HYMN"
40002 DATA"CHANT","CHORD","VOCAL","CALANDO","CLEF"
,"STAFF","DOT","FORTE"
40003 DATA"FORTISSIMO","HARMONIC","LEGATO","LOCO",
"METRONOME","MORDENT"
40004 DATA"SHARP","FLAT","NATURAL","OPUS","PEDAL",
"PIANISSIMO","PAUSE"
40005 DATA"PIZZICATO","QUARTETTE","QUINTETTE","QUI
NTUPLET","SEGUE","SEPTET"
40006 DATA"SEXTET","SEMITONE","STACCATO","TREMOLO"
,"TRILLO","TRIPLET","SOLO"
40007 DATA"VIBRATO":REM 50 80 FAR
40010 DATA"GUITAR","PIANO","VIOLIN","TRUMPET","SAX
OPHONE","HARPSICHORD"
40011 DATA"ORGAN","PIPES","FLUTE","ACCORDION","DRU
MS","BASS","HARP","WHISTLE"
40012 DATA"KAZOO","CYMBAL","SNARE","FIDDLE","BOW",
"XYLOPHONE","TUBA","BAGPIPES"
40013 DATA"CLARINET","TROMBONE","VIBES","HARMONIUM
","ALPHORN","PANTALEON"
40014 DATA"ORCHESTRION","VIOLA","PICCOLO","BASSOON
","SYNTHESISER","MOOG"
40015 DATA"HARMONICA","GONG","BOTTLENECK","SLIDE",
"STEEL","ACOUSTIC"
40016 DATA"SPANISH","ELECTRIC","TUBULAR","BELLS","
MANDOLIN","BANJO","UKELELE"
40017 DATA"PIANOLA","WURLITZER","WASHBOARD":REM 10
0 80 FAR
40018 DATA"STRING","TUNING","FRET","MAJOR","MINOR"
,"TONE","TREBLE","SEMIBREVE"
```

```
40019 DATA"SEMIQUAVER","PITCH","OCTAVE","BEAT","RH
YTHM","MELODY","TUNE","SCALE"
40020 DATA"KEYS","HARMONY","ADANTE","MODERATO","CO
NTRALTO","ALTO","TENOR"
40021 DATA"SOPRANO","ALLEGRETTO","TREMOLO","CELLO"
,"REED","WIND","RECORDER"
40022 DATA"BUGLE","HORN","BAZOOKA","LYRE","LUTE","
UKULELE","BALALAIKA","DUET"
40023 DATA"PRESLEY","TRIO","BAND","GROUP","BEATLES
","DYLAN","FELICIANO"
40024 DATA"MANTOVANI","SINATRA","PERCUSSION","VOCA
LS","WIND","TIMPANI"
40025 DATA"FOLK","BLUES","BALLAD","COUNTRY","ROCK"
,"CLASSICAL","HAYDN"
40026 DATA"BRAHMS","LISZT","OPERA","BEETHOVEN","BA
CH","WAGNER","HANDEL"
40027 DATA"WALTZ","SONG","VOICE","SCHUBERT","MOZAR
T","BARRYMANILOW"
62000 V=53248:POKEV+23,0:POKEV+29,0
62002 FORI=0TO15:POKEV+I,0:NEXT
62005 B(0)=248:B(1)=249:B(2)=250:B(3)=251:B(4)=252
:B(5)=253:B(6)=254:B(7)=255
62010 NS=T:IFNS=0THENRETURN
62015 FORA=1TONS
62020 READSK,M1,M2:IFSK=0THENPOKEV+28,PEEK(V+28)AN
D255-2^(A-1):GOTO62030
62025 POKEV+28,PEEK(V+28)OR2^(A-1):POKEV+37,M1:POK
EV+38,M2
62030 READCO:POKEV+38+A,CO:POKE2039+A,B(A-1)
62035 FORC=B(A-1)*64TOB(A-1)*64+63:READQ:POKEC,Q:N
EXT:NEXT:RETURN
63000 S=54272:POKES+5,9:POKES+6,100:POKES+3,0:POKE
S+2,255:POKES+4,65
63002 RETURN

READY.
```

## Explanation

Line 5 : Set a black border and black background, set everything into upper and lower case instead of upper case and graphics, clear the screen, turn the print colour to yellow, and print up the title of the program.

Lines 15-45 : On screen instructions and a little bit about the game.

Line 50 : Define T to equal 8, which tells one of the later routines that

we want to define 8 sprites. Go to the routine at 62000, which defines the sprites, and then go to the routine at 63000, which defines voice one to be ... we'll see what later.

Lines 55-65 : Press space to continue message, wait until the space bar is pressed, then get ready to read all the word data in.

Line 70 : Dimension the array W$ to contain 171 elements (172 actually, I know, but we haven't bothered with element zero), since we're going to be reading in 171 words. Obviously this would have to be changed if you wanted to add to the word database yourself. Then define a string G$ to contain 26 elements, since this string is going to handle all your alphabetic input: there are 26 letters in the alphabet, so we reserve one space for each of them. The word data is then read in in the last part of the line.

Line 74 : Clear out G$, and set H$ equal to nothing. H$ will be used to print out your guess each time and show your progress, or lack of it.

Line 75 : Clear the screen, print the title up in reverse field, and set the volume to maximum.

Line 80 : Select a random number between 1 and 171, pick out that word from our word database by choosing W$(J), and finally set G, the number of guesses, to 1.

Line 82 : Since the data was entered as upper case first letter, lower case for the rest of the word, switch the first letter to lower case as well by taking the ASCII value of it and subtracting 128, and adding it to the original word minus the first letter to make a complete word again. Phew!

Line 84 : Perform a loop the same number of times as there are letters in the word, and set up our display string accordingly.

Line 85 : Inform the player how many letters are in the word.

Line 90 : Start of display to show which letters have been used.

Lines 95-120 : Go through each element of the array G$ to see if it has a letter in it. If it has, print it up on the screen, if it hasn't then just go on to the next line.

Line 122 : If your answer string H$ is equal to the selected word W$ then you've obviously cheated the hangman, so print up a message

of congratulation and trot off to line 2000 to play a cheery tune, among other things.

Line 125 : Tell the player to enter his guess.

Line 130 : Print up the guess string.

Line 135 : Clear the keyboard buffer.

Line 136 : Turn the waveform for voice one off.

Line 140 : Get a character.

Line 142 : POKE a couple of values into the high and low order note values for voice one. The note gets higher the more guesses the player takes.

Line 145 : Reject anything that isn't an alphabetic character.

Line 146 : Turn the waveform for voice one on.

Line 150 : Set the Gth element of our letters guessed string to contain the letter just typed in, and increment G by one.

Line 155 : Start of loop to check player input.

Line 160 : If the input matches a letter in our chosen word, then off to 500.

Line 165 : Rest of the loop.

Line 170 : A match hasn't been found, so go to line 600 to change the screen display.

Line 175 : Reset the variable Z ready for the next input, and back to line 85 for more.

Line 500 : Build up the guess string H$ according to what has just been typed in.

Line 502 : Increment the counter Z to show that a letter match has been found.

Line 510 : Back to line 165 for more checks.

Lines 600-604 : Increment ZZ counter, which goes up when a letter match has not been found, and turn on all sprites, together with expanding them in all directions.

Line 606 : Go to appropriate routine for updating screen display.

Lines 610-680 : Turn various sprites on and off as suitable, and position them on the screen.

Lines 690-700 : The player's blown it, so go to the routine at 1000 to play the death march. Then put a silly sprite display on screen just to rub in the player's failure.

Line 704 : A delay loop.

Lines 705-740 : Check to see if the player wants another game.

Lines 1000-1006 : Build up string to play the death march, and go to the routine at 11000 which handles playing it.

Lines 2000-2004 : Build up tune to celebrate success (a crack-throated rendition of 'The Sun Has Got His Hat On'), and then go to line 11000 to play it.

Lines 11000-11032 : Playing the tune based on the string Z$. The routine essentially searches through for commas (line 11006), converts the parts of the string that are not commas into numerical values (line 11007), and then plays those values in line 11032. An interesting exercise, if you're getting your data confused and wish to use strings instead.

Lines 30000-30079 : Sprite data.

Lines 40000-40027 : Word base. Sorry about the last entry, it wasn't my idea!

Lines 62000-62035 : Read in sprite data and store it at appropriate locations. Multi-colour sprites as well, so set everything accordingly.

Lines 63000-63002 : Set ADSR parameters for voice one, along with pulse width, before selecting pulse waveform.

A lengthy program, but it incorporates some interesting programming techniques.

# 7
# Getting Tuneful

## Introduction

In order to play a few tunes on the 64, it would be an easy matter to get you to turn to Chapter 10, type in the listing given there, and start making up your own tunes. However, we are not all as musical as we would like to be, and certainly not all of us could sit down in front of a keyboard and produce recognisable tunes.

The program in Chapter 10 will certainly help you to produce some interesting noises if your skill as a musician is on a par with mine i.e. zero. On the other hand, it would be nice actually to learn something about music as well as just making a noise, and so over the course of the next three chapters we're going to match musical theory on the 64 with some musical practice programs, designed to increase your knowledge of music generally, and give you a feel for what note lies where on the keyboard, how the position of a musical note on the stave relates to the noise that note produces, and so on.

Towards the end of this book there are a couple of sample tunes, made famous originally by the Beatles, so everyone should (I hope!) recognise them. With the title of this book being what it is, one tune was a fairly obvious choice, and the other I happened to hear someone whistling the other day. Anyway, it's an easy tune to type in.

### Producing tunes

Although both tunes are familiar ones, I could certainly not have simply sat down at the keyboard and typed them in. If you can are capable of doing that, fine. If not, perhaps a few words would be in order.

The easiest way to commit some tunes to tape or disk is to pay a visit to your local music store and get a few song sheets. Wise Publications, distributed by Music Sales Limited, produce a number of excellent books for the beginner, including a very good series of '101 ... Songs

for Buskers'. These titles cover Beatles' songs, general songs, anybody under the sun's songs, and they're pretty cheap as well. Worth searching out.

There are many other books of songs, as any good musical store will tell you. For the sake of a couple of quid, it's worth acquiring one of these books and then you can at least annoy the relatives with something vaguely tuneful, rather than subjecting them to a barrage of electronic noise any time they come near you and your computer.

## Musical tunes

We've already introduced a couple of methods of playing tunes, using the string technique in the 'Musical Hangman' program, and the data technique in the 'Three-Part Harmony?' listing. There are, of course, many other methods by which one could string together a collection of notes and repeat them ad infinitum. We'll look at just one method, albeit one that is commonly used in the top games programs around today, in Chapter 13.

For now, a musical listing to give you a little bit of fun, as well as a better ear for music. This is a musical equivalent of the Simon game, which in its earlier form consisted of what looked rather like a dustbin lid that had been divided up into four coloured portions. One or more of these portions would light up in turn, and the player had to hit, in the correct sequential order, those portions that had previously been lit up.

In this musical variant, the computer will play a series of notes (four at first, and then gradually more as your skill in repeating musical notes grows greater). These notes will be played on a representation of a piano keyboard on the screen. You get the opportunity to practise on that keyboard before the game itself starts.

Having heard the notes and seen where they were played, you then have to repeat them in the exact order in which they appeared. Get it wrong, and it's back to the beginning again, but get it right and you'll be presented with ever more notes until your memory cracks under the strain!

## Musical Simon

Before you set about typing in the listing, a few words of warning.

The complicated part in lines 10 to 110 is not as bad as it seems, since a lot of the lines are duplicates of each other. For instance, line 20, 30 and 40 are the same as line 10 (although for the purposes of this listing they've had to be split up into two lines each).

Again, lines 70, 80, 90 and 100 are all the same. In both these instances just type in the first line and use the 64's editing keys to repeat the line, to save you typing it out again and again. If you're getting a little bit confused over what all the symbols mean, refer back to Chapter 1 where they were all explained in depth. If it's any help, what you're trying to draw is a piano keyboard.

Bear in mind that we'll be using this keyboard layout in a couple of games, so when you've typed it in once, it will be used again.

```
5 CO=4:GOSUB30000:GOSUB10000
9 GOTO200
10 PRINT"[8SP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF,SP]":
11 PRINT"[RVS,2SP,OFF,4SP,OFF,2SP,OFF]"
12 REMLINES 10 AND 11 COMBINE TO FORM ONE LINE WHE
N PRINTED ON THE 64
20 PRINT"[8SP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF,SP]":
21 PRINT"[RVS,2SP,OFF,4SP,OFF,2SP,OFF]"
22 REMLINES 20 AND 21 COMBINE TO FORM ONE LINE WHE
N PRINTED ON THE 64
30 PRINT"[8SP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF,SP]":
31 PRINT"[RVS,2SP,OFF,4SP,OFF,2SP,OFF]"
32 REMLINES 30 AND 31 COMBINE TO FORM ONE LINE WHE
N PRINTED ON THE 64
40 PRINT"[8SP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF,SP]":
41 PRINT"[RVS,2SP,OFF,4SP,OFF,2SP,OFF]"
42 REMLINES 40 AND 41 COMBINE TO FORM ONE LINE WHE
N PRINTED ON THE 64
50 PRINT"[6SP,RVS,WHT,2SP,BLK,2SP,WHT,SP,BLK,WSP,W
HT,SP,CBMM,2SP,BLK,2SP,WHT,]":
51 PRINT"[SP,BLK,SP,WHT,SP,BLK,2SP,WHT,SP,CBMM,2SP
,BLK,2SP]"
52 REM LINES 50 AND 51 COMBINE TO FORM ONE LINE WH
EN PRINTED ON THE 64
60 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]":
61 PRINT"[2SP,CBMM,2SP,CBMM]"
```

```
62 REM LINES 60 AND 61 COMBINE TO FORM ONE LINE WH
EN  PRINTED ON THE 64
70 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]";
71 PRINT"[2SP,CBMM,2SP,CBMM]"
72 REM LINES 70 AND 71 COMBINE TO FORM ONE LINE WH
EN  PRINTED ON THE 64
80 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]";
81 PRINT"[2SP,CBMM,2SP,CBMM]"
82 REM LINES 80 AND 81 COMBINE TO FORM ONE LINE WH
EN  PRINTED ON THE 64
90 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]";
91 PRINT"[2SP,CBMM,2SP,CBMM]"
92 REM LINES 90 AND 91 COMBINE TO FORM ONE LINE WH
EN  PRINTED ON THE 64
100 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,
2SP,CBMM,2SP,CBMM,2SP,CBMM]";
101 PRINT"[2SP,CBMM,2SP,CBMM]"
102 REM LINES 100 AND 101 COMBINE TO FORM ONE LINE
 WHEN  PRINTED ON THE 64
110 RETURN
200 PRINT"[CLR]";CO;" NOTES COMING UP![4CD]":GOSUB
10
201 PRINT"[HOME,3CD,WHT]          #  #     #  #  #
    #"
202 PRINT"[HOME,4CD,WHT]         C  D     F  G  A
   C"
203 PRINT"[HOME,15CD]       C  D  E  F  G  A  B  C
"
204 FORI=1TO1000:NEXTI:FORI=1TOCO:KE$(I)=MID$(KE$,
INT(RND(.5)*13+1),1):NEXTI
206 FORJ=1TOCO
207 POKEV+4,0
208 FORI=1TO14:IFKE$(J)=MID$(KE$,I,1)THENGOSUB5000
0:I=14
209 NEXTI:POKEV+4,65:FORK=1TO500:NEXTK,J
210 PRINT"[4CD]NOW IT'S YOUR TURN!"
215 A=1
220 P=0:GETN$:IFN$=""THEN220
225 FORI=1TO14:IFN$=MID$(KE$,I,1)THENP=1
226 NEXTI:IFP=0THEN220
230 AN$(A)=N$
232 FORI=1TOLEN(KE$)
234 IFN$=MID$(KE$,I,1)THENPOKEV+4,0:POKEV+4,65:GOS
UB50000
235 NEXT
240 A=A+1:IFA>COTHEN250
245 GOTO220
250 FORI=1TOCO
260 IFAN$(I)<>KE$(I)THEN300
```

```
270 NEXT
280 PRINT"[2CD]CORRECT!":FORJ=0TO1000:NEXT
285 ZQ=ZQ+1:IFZQ=4THENCO=CO+1:ZQ=0
286 GOTO200
300 PRINT"[2CD,RVS]WRONG!  ":FORJ=0TO1000:NEXT
310 PRINT"YOU SHOULD HAVE PLAYED:"
312 A=1
314 POKEV+4,0
316 FORI=1TO14:IFKE$(A)=MID$(KE$,I,1)THENGOSUB5000
0:I=13
318 NEXTI:POKEV+4,65:FORK=1TO500:NEXTK
319 A=A+1:IFA<=COTHEN314
320 PRINT"[CLR,2CD]YOU FAILED TRYING TO REPEAT ":C
O:" NOTES."
321 PRINT"[2CD]ANOTHER GO (Y OR N)?":POKES+1,0:POK
ES,0
330 GETG$:IFG$="Y"THENCO=4:GOTO200
340 IFG$="N"THENPRINT"[2CD]BYE.":END
350 GOTO330
9999 END
10000 PO53281,2:POKE53280,6:POKE53272,23:PRINT"[CL
R,WHT,RVS]WELCOME TO MUSICAL SIMON"
10002 PRINT"[CD]THIS IS A MUSICAL VERSION OF THE P
OPULARGAME OF SIMON."
10004 PRINT"[CD]I WILL PLAY A TUNE. AND INDICATE W
HICH  NOTES THE TUNE ":
10006 PRINT"CONSISTS OF.  YOU MUST   THEN REPEAT T
HE TUNE EXACTLY."
10008 PRINT"[2CD]":GOSUB10:GOSUB20000
10010 PRINT"[CLR]THE FIRST TUNE WILL ONLY HAVE A F
EW     NOTES IN IT, BUT AFTER ":
10012 PRINT"EVERY FOURTH     SUCCESSFUL  REPEAT, T
HE TUNE WILL GROW  BY ONE NOTE."
10014 PRINT"[2CD]GET JUST ONE NOTE WRONG. AND YOU'
LL BE  GIVEN YOUR SCORE. THE ":
10016 PRINT"TUNE WILL THEN RE-SET TO ITS STARTING
LENGTH OF 4 NOTES IFYOU ":
10018 PRINT"WISH TO HAVE ANOTHER GO."
10020 PRINT"[2CD]PLAY USING THE KEYS INDICATED."
10022 GOSUB20000
10024 PRINT"[CLR,4CD]":GOSUB10
10026 PRINT"[HOME,4CD,WHT]         W  E     T  Y  U
     O"
10028 PRINT"[HOME,15CD]       A  S  D  F  G  H  J
 K"
10030 PRINT"[2CD]TRY IT NOW. JUST TO GET THE FEEL
OF THE KEYBOARD.  WHEN YOU'RE FED":
10032 PRINT" UP, PRESS    'SPACE' TO GET THE SHOW
ROLLING."
10040 GETA$:IFA$=""THEN10040
10045 IFA$=" "THEN10999
10050 FORI=1TOLEN(KE$)
```

```
10060 IFA$=MID$(KE$,I,1)THENPOKEV+4,0:POKEV+4,65:G
OSUB50000
10070 NEXT
10090 GOTO10040
10999 POKES+1,0:POKES,0:RETURN
20000 PRINT"[2CD,RVS]PRESS 'SPACE' TO CONTINUE."
20002 FORI=1TO10:GETSP$:NEXT
20004 GETSP$:IFSP$<>" "THEN20004
20006 RETURN
30000 V=54272:POKEV+3,255:POKEV+5,9:POKEV+6,0:POKE
V+4,65:POKEV+24,15
30001 DIMNO(28),KE$(255),AN$(255),SP(28)
30002 DATA4,73,4,208,5,103,5,185,6,108,7,53,8,23,8
,147,4,139,5,25,6,16,6,206
30004 DATA7,163,9,21
30006 FORI=1TO28:READNO(I):NEXT
30008 KE$="ASDFGHJKWETYUO"
30010 S=53248
30012 FORI=0TO62:READA:POKE832+I,A:NEXT
30014 POKE2040,13:POKES+23,0:POKES+29,0:POKES+28,0
:POKE8+39,5:X=0:Y=0
30015 POKES+1,Y:POKES,X:POKES+21,1
30016 FORI=1TO28:READSP(I):NEXT
30018 RETURN
40001 DATA0,0,0,0,0,0,0
40002 DATA32,0,0,48,0,0,56
40003 DATA0,0,60,0,0,38,0
40004 DATA0,34,0,0,34,0,0
40005 DATA36,0,0,32,0,0,32
40006 DATA0,0,32,0,0,32,0
40007 DATA3,224,0,7,224,0,15
40008 DATA240,0,15,240,0,7,224
40009 DATA0,1,128,0,0,0,0
41000 DATA 73,135,97,135,121,135,145,135,169,135,1
93,135,217,135,241,135
41002 DATA 86,95,110,95,158,95,182,95,206,95,255,9
5
50000 POKEV+1,NO(I+I-1):POKEV,NO(I+I):POKES+1,SP(I
+I):POKES,SP(I+I-1):RETURN
```

### Explanation

As usual, our dissection of the program starts with the very first line.

Line 5 : Set the variable CO to equal four. This determines how many notes will be played for the player to attempt to repeat. Then go to the routine at 30000 to set up some data, and line 10000 to start off the introduction to the game.

Line 9 : Off to line 200, which is the start of the game.

Lines 10-110 : Set up the piano keyboard display, as we discussed earlier.

Line 200 : Clear the screen, inform the player how many notes he's going to have to remember, and go to the routine at line 10 to draw the keyboard.

Lines 201-203 : Display which key is playing which note.

Line 204 : After a slight delay, set up CO random notes, taken from the letters in the string KE$ defined earlier on in a subroutine at the end of the program.

Line 206 : Set up a loop to be repeated CO times.

Line 207 : Switch the waveform of voice one to zero.

Line 208 : Start of the loop to identify which random notes to play. The routine at line 50000 plays the notes, and also moves our sprite around on screen to show the player which note has in fact been played, as well as allowing him to listen to it.

Line 209 : Finish off the loop.

Line 210 : Inform the player that it's his turn.

Line 215 : Set the variable A to equal one. This will check to see how many notes have been played.

Line 220 : Set a flag to equal zero, and wait for a key to be pressed.

Line 225 : Set up a loop to try to identify which key has been pressed. If it is one of the keys that will produce a musical note, set the flag to equal one.

Line 226 : Otherwise continue around the loop, and if the key isn't a recognised one go back and try again.

Line 230 : The Ath key pressed by the user is set to equal the last key pressed.

Line 232 : Start a loop to check for every playable note.

Line 234 : If the key is recognised, then turn the waveform for voice one off, turn it on again, and go to the routine at 50000 to play the note and move the sprite around.

Line 235 : Next step round the loop.

Line 240 : Check to see if enough keys have been pressed. If they have then start checking to see whether the player's attempt at repeating the notes in order was a good one.

Line 245 : We haven't had enough notes yet, so go back and get some more.

Line 250 : Start off loop to check every note played by the player.

Line 260 : If the Ith key pressed doesn't equal the Ith random note generated earlier, then a mistake has been made, so go to the routine at 300 to mention this.

Line 270 : Next step around the loop.

Line 280 : With a slight delay for effect, inform the player that he's got it all right.

Line 285 : Check for incrementing the number of notes to be repeated. After every fourth go, this is incremented by one.

Line 286 : Back to line 200 for another go.

Line 300 : With another dramatic pause, the player is informed of his error.

Line 310 : Start of routine to inform the player what he should have played.

Line 312 : Reset variable A to 1.

Line 314 : Turn the waveform off.

Line 316 : Check our randomly generated set of notes, and if one is found then go to the routine at 50000 to play it.

Line 318 : Next step around the loop. Turn the waveform on, and delay for a short while (about half a second).

Line 319 : All notes played? If not, go back and play another one.

Line 320 : Inform the player of his demise.

Lines 321-350 : Check to see if the player wants to have another go.

Line 9999 : Early line left in from days of checking program!

Lines 10000-10040 : Instructions, and invite the player to have a go on the practice keyboard.

Lines 10040-10999 : Let the player play some practice notes until he gets fed up and presses the space bar.

Lines 20000-20006 : Routine for waiting until space bar is pressed.

Line 30000 : Set sound variables, including ADSR sequence and waveform.

Line 30001 : Initialise and declare a few variables.

Line 30002-30004 : Data for musical notes according to which key is pressed.

Line 30006 : Read the note data in.

Line 30008 : Declare KE$ to equal every valid key that can be pressed.

Line 30010 : Set variable S to equal start of sprite memory.

Line 30012 : Read sprite data and store it in the 13th block for sprite data.

Line 30014 : Inform the computer of this fact, and have an un-enlarged green sprite which will start life in the top left hand corner of the screen.

Line 30015 : Turn the sprite on, but display it off screen.

Line 30016 : Read data for sprite position on screen corresponding to various key presses.

Line 30018 : Guess what?

Lines 40000-40009 : Sprite data.

Lines 40010-40012 : Sprite screen position data corresponding to keys in KE$.

Line 50000 : Routine to play note and move sprite to correct position on screen.

As in all programs of this type, it is not only the program itself that is important, but also the collection of short routines contained within it that perform various important operations. Look and learn, as they say, and as with all programs in this book please feel free to modify them as you see fit.

## A non-musical interlude

In the midst of the wealth of musical programs covered in this book, one or two of the routines used were inspired by some other, non-musical program. Since it would seem a shame to leave out these undeniably useful programs, some of them have been included here. This one was written by R. Crothers, and allows you to save any portion of memory to tape or disk.

This seemed like a reasonable enough program, and a very useful utility to have. So often, when typing in a machine code program without a monitor to assist you, it becomes impossible to save that program later. Using this, that is no longer a problem. It was used, in conjunction with the synthesiser program coming up later, to store some background musical tunes on to disk, for later recall within the main synthesiser program.

## Memory saver

Type in the program as shown, and save it before running it, just in case. If all is well, the screen should clear to reveal the name of the program, and a request to enter a filename. Following on from this, you are asked to enter a device to save your program on to (tape or disk), and then the routine is initialised with a SYS679 call when necessary.

```
3 POKE53281,0:POKE53280,9:POKE646,2:PRINT"[CLR,YEL
]"
4 PRINTTAB(11)"[RVS]    SAVER  64      "
5 DATA169,54,133,1
10 DATA169,8,162,8,160,255,32,186,255
20 DATA169,2,162,211,160,2,32,189,255
30 DATA169,1,133,247,169,8,133,248,162,255,160,127
,169,247,32,216,255
35 DATA169,55,133,1,96
40 FORI=679TO722:READA:POKEI,A:NEXT
50 N$="":DEV=1:S=2049:E=32767
60 INPUT"[CD]FILE NAME     ":N$:N$=LEFT$(N$,16)
62 IFN$=""THEN60
70 INPUT"DEVICE (T/D) ":DEV$
71 IFDEV$="T"THENDEV=1:GOTO80
72 IFDEV$="D"THENDEV=8:GOTO80
74 GOTO70
80 S=49152:REM START ADDRESS ($C000 IN  THIS EXAMP
LE)
90 E=53247:REM END ADDRESS )$CFFF IN THIS EXAMPLE)
100 K=S:GOSUB1000:POKE702,A1:POKE706,A2
110 K=E:GOSUB1000:POKE710,A1:POKE712,A2
120 SA=255:POKE688,SA
130 K=LEN(N$):POKE693,K
140 FORI=1TOK
150 POKE722+I,ASC(MID$(N$,I,1)):NEXT
155 POKE684,DEV:POKE686,DEV
156 PRINT"[CD]LOAD IN MACHINE CODE &"
160 PRINT"TYPE 'SYS 679' TO SAVE":END
1000 A2=INT(K/256):A1=K-A2*256:RETURN
```

**Explanation**

Line 3 : We'll have a black border and background, followed by clearing the screen and setting the printing colour to yellow, thus overriding the POKE 646,2 earlier on in the line.

Line 4 : So this is what the program's really called.

Lines 5-30 : Machine code data. Enter these numbers very carefully, as some of them are calls to internal 64 routines, and we don't want to end up in the wrong part of ROM at the wrong time.

Line 40 : Read the data in and POKE it into place.

Line 50 : Set some default values for the file name, device number, and start and end addresses of the machine code to be saved.

Line 60 : Input the file name, and take the first sixteen characters of it in case anyone types in a ludicrously long name.

Line 62 : You can't just press return and expect to get away with it.

Line 70 : Input the device (tape or disk).

Line 71 : If the device is tape, then the device number is 1, and go to line 80.

Line 72 : Ditto for disk.

Line 74 : No strange letters typed into this program!

Line 80 : Set start address of machine code routine to be saved.

Line 90 : And set the end address as well. Sorry about the bracket being the wrong way around.

Lines 100-110 : POKE high and low order values for start and end addresses into locations 702,706,710 and 712.

Line 120 : POKE688 with 255 decimal, or $FF hexadecimal.

Line 130 : Put the length of the file name into location 693.

Lines 140-150 : Put the ASCII codes for the filename into locations 722 onwards.

Line 155 : POKE the device number into locations 684 and 686.

Lines 156-160 : Over and out with a simple message.

# 8
# Ring Modulation and Synchronisation

## Introduction

When it comes to more advanced techniques, the Commodore 64 scores over most of the other home computers currently available. True, these techniques aren't exactly easy to implement on the 64, but at least they are in there somewhere, and with a little bit of time and effort you can produce some very interesting results indeed.

We've already seen how altering the pulse width can affect the quality of the note being produced. However, we can't use pulse widths all the time, since the ring modulation effect requires us to use a triangle waveform.

What precisely is ring modulation? Well, it's a fancy way of saying that one voice is going to affect another voice in a particular way, and since words are usually inadequate to describe a musical effect, let's take a look at a short program.

```
10 V=54272
20 POKEV+4,0:POKEV+5,9:POKEV+6,0:POKEV+4,33
30 POKEV+18,0:POKEV+19,9:POKEV+20,0:POKEV+17,0
40 POKEV+16,255:POKEV+18,65
50 POKEV+24,15
60 A=INT(RND(.5)*20+10):B=B+1:IFB>255THENB=0
70 POKEV+1,A:POKEV+15,B
80 FORI=1TO1000:NEXT:GOTO20
```

All this program will do as it stands is to produce two different notes for voice one and voice three. Here, voice one is currently a sawtooth waveform, and voice three is a pulse waveform. However, if we alter lines 20 and 50 as shown below, you should be able to hear the difference.

```
20 POKEV+4,0:POKEV+5,9:POKEV+6,0:POKEV+4,17
50 POKEV+24,17
```

This is now modulating voice one with respect to voice three, in other words, we are now combining the frequencies of voices one and three. Not a very good demonstration, but at least it works. The synthesiser program in Chapter 10 will give you a far better idea of what is happening with ring modulation.

Synchronisation involves taking the note produced by voice one and synchronising it with the note produced by voice three. For this effect to work properly, voice three has got to be set to a frequency lower than that of voice one. Rather than give a short demonstration here, we'll wait for the synthesiser program before going into this in more detail, as synchronisation works better with background rhythms as opposed to single notes.

## Musical practice

Continuing with our policy of presenting educational musical programs, this one allows you to play a series of musical notes and see those notes depicted on a stave above the keyboard as you play. This gives some feeling for what a note will sound like when it is placed in a particular spot on the stave, and, interestingly enough, seems to work better in reverse. In other words, you are playing notes and making those notes appear at a particular place on the stave, but what you seem to learn is how a note will sound when it is displayed on the stave. At least it improved my sight reading.

```
0 POKE53248+21,0
1 POKE53281,11:POKE53280,11:POKE53272,23:PRINT"[CL
R,BLK,RVS]NOTE PRACTICE."
5 GOSUB40000:GOSUB50000:PRINT"[CLR,2CD]":GOSUB6:GO
TO110
6 PRINT"[BLK,3SP,6SHIFT*,CBMR,23SHIFT*]
7 PRINT"[BLK,3SP,6SHIFT*,SHIFT+,23SHIFT*]
8 PRINT"[BLK,3SP,6SHIFT*,SHIFT+,23SHIFT*]
9 PRINT"[BLK,3SP,6SHIFT*,SHIFT+,23SHIFT*]
10 PRINT"[BLK,3SP,6SHIFT*,CBME,23SHIFT*]
11 POKEV+23,3:POKEV+1,70:POKEV,40:POKEV+39,0
12 POKEV+2,60:POKEV+3,70:POKEV+40,0
13 POKEV+41,8:FORI=42TO46:POKEV+I,1:NEXT
18 PRINT"[2CD]
19 PRINT"[4SP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF,SP]";
```

```
20 PRINT"[RVS,2SP,OFF,4SP,RVS,2SP,OFF,SP,RVS,2SP,O
FF]"
21 REM LINES 19 AND 20 FORM ONE LINE TOGETHER ON T
HE 64
22 PRINT"[4SP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF,SP]";
23 PRINT"[RVS,2SP,OFF,4SP,RVS,2SP,OFF,SP,RVS,2SP,O
FF]"
24 REM LINES 22 AND 23 FORM ONE LINE TOGETHER ON T
HE 64
30 PRINT"[4SP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF,SP]";
31 PRINT"[RVS,2SP,OFF,4SP,RVS,2SP,OFF,SP,RVS,2SP,O
FF]"
32 REM LINES 31 AND 32 FORM ONE LINE TOGETHER ON T
HE 64
40 PRINT"[4SP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF,SP]";
41 PRINT"[RVS,2SP,OFF,4SP,RVS,2SP,OFF,SP,RVS,2SP,O
FF]"
42 REM LINES 41 AND 42 FORM ONE LINE TOGETHER ON T
HE 64
50 PRINT"[2SP,RVS,WHT,2SP,BLK,2SP,WHT,SP,BLK,2SP,W
HT,SP,CBMM,2SP,BLK,2SP,WHT]";
51 PRINT"[SP,BLK,2SP,WHT,SP,BLK,2SP,WHT,SP,CBMM,2S
P,BLK,2SP,WHT,SP,BLK,2SP,WHT,SP]";
52 PRINT"[CBMM]":REM LINES50,51 AND 52 COMBINE TOG
ETHER TO FORM ONE LINE
60 PRINT"[2SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]";
61 PRINT"[2SP,CBMM,2SP,CBMM,2SP,CBMM,2SP,CBMM]"
62 REMLINES 60 AND 61 FORM ONE CONTINUOUS LINE ON
THE 64
70 PRINT"[2SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]";
71 PRINT"[2SP,CBMM,2SP,CBMM,2SP,CBMM,2SP,CBMM]"
72 REMLINES 70 AND 71 FORM ONE CONTINUOUS LINE ON
THE 64
80 PRINT"[2SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]";
81 PRINT"[2SP,CBMM,2SP,CBMM,2SP,CBMM,2SP,CBMM]"
82 REMLINES 80 AND 81 FORM ONE CONTINUOUS LINE ON
THE 64
90 PRINT"[2SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]";
91 PRINT"[2SP,CBMM,2SP,CBMM,2SP,CBMM,2SP,CBMM]"
92 REMLINES 90 AND 91 FORM ONE CONTINUOUS LINE ON
THE 64
100 PRINT"[2SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,
2SP,CBMM,2SP,CBMM,2SP,CBMM]";
101 PRINT"[2SP,CBMM,2SP,CBMM,2SP,CBMM,2SP,CBMM]"
102 REMLINES 100 AND 101 FORM ONE CONTINUOUS LINE
```

```
ON THE 64
105 RETURN
110 PRINT"[CD.BLK]PRESS 'SPACE' TO CONTINUE."
120 FORI=1TO10:GETSP$:NEXT
130 GETSP$:IFSP$<>" "THEN130
135 POKEV+21,0
140 PRINT"[CLR,BLK]THIS PROGRAM AIMS TO TEACH YOU
HOW TO   RECOGNISE NOTES AS THEY ";
150 PRINT"APPEAR ON A     MUSICAL STAVE.  AS YOU P
LAY USING THE   KEYBOARD,";
155 PRINT" THE NOTES APPEAR ON THE STAVE ABOVE.
160 PRINT"[CD]THE KEYBOARD IS PLAYED USING THE SIM
ILARKEYS TO THE MUSICAL SIMON";
170 PRINT" GAME: TRY IT  NOW, USING KEYS A-K AND W
,E,T,Y,U,O :    '!' TO QUIT."
180 PRINT"[4CU]":GOSUB18
182 POKEV+21,4:POKEV+41,8:GOSUB185:GOTO230
185 GETA$:IFA$=""THEN185
186 IFA$="!"THENPOKEV+21,0:RETURN
190 FORI=1TO17:IFA$=MID$(KE$,I,1)THEN200
195 NEXT
196 GOTO185
200 POKES+4,0:POKES+4,65
210 POKES+1,NO(I+I-1):POKES,NO(I+I):POKEV+4,SP(I+I
-1):POKEV+5,SP(I+I)
220 GOTO195
230 PRINT"[CLR,BLK]NOTE PRACTICE.[2CD]":GOSUB6:POK
EV+21,255
250 FORI=1TO5:POKEV+4+I*2,100+I*30:NEXT
256 PRINT"[2CD]PLAY AWAY:  '!' TO QUIT."
257 REM KEY PRESSED
258 FORJ=1TO5
260 GETA$:IFA$=""THEN260
261 IFA$="!"THENPOKEV+21,0:PRINT"[CLR,2CD,BLK]BYE.
":END
262 IFASC(A$)<58ORASC(A$)>90THEN260
264 A=ASC(A$)
268 FORI=1TO17:IFA$=MID$(KE$,I,1)THEN274
270 NEXTI:GOTO260
272 REM PLAY NOTE
274 POKES+4,0:POKES+4,65
276 POKES+1,NO(I+I-1):POKE8,NO(I+I):POKEV+4,SP(I+I
-1):POKEV+5,SP(I+I)
277 POKEV+5+J*2,NP(I)
278 IFI>10THENPOKE2042+J,255:GOTO280
279 POKE2042+J,254
280 NEXTJ:GOTO258
30001 DATA0,0,0,0,8,0,0
30002 DATA34,0,0,66,0,0,66
30003 DATA0,0,66,0,0,68,0
30004 DATA0,88,0,0,224,0,3
30005 DATA32,0,4,124,0,8,162
```

```
30006 DATA0,9,49,0,9,17,0
30007 DATA8,146,0,4,20,0,3
30008 DATA248,0,0,16,0,12,16
30009 DATA0,12,48,0,7,192,0,0
30011 DATA0,0,0,0,0,0,0
30012 DATA0,0,0,0,0,0,0
30013 DATA0,0,0,0,0,24,0
30014 DATA0,126,0,0,153,0,1
30015 DATA30,0,1,24,0,1,24
30016 DATA0,1,24,0,0,154,0
30017 DATA0,124,0,0,24,0,0
30018 DATA0,0,0,0,0,0,0
30019 DATA0,0,0,0,0,0,0,0
30021 DATA0,0,0,0,0,0,0
30022 DATA24,0,0,24,0,0,28
30023 DATA0,0,28,0,0,22,0
30024 DATA0,19,0,0,17,128,0
30025 DATA16,128,0,16,128,0,16
30026 DATA0,0,16,0,0,16,0
30027 DATA0,16,0,0,16,0,7
30028 DATA144,0,15,224,0,15,192
30029 DATA0,7,128,0,0,0,0,0
30031 DATA0,0,0,0,0,0,0
30032 DATA24,0,0,24,0,0,28
30033 DATA0,0,28,0,0,22,0
30034 DATA0,19,0,0,17,128,0
30035 DATA16,128,0,16,128,0,16
30036 DATA0,0,16,0,0,16,0
30037 DATA0,16,0,0,16,0,7
30038 DATA144,0,15,224,0,15,192
30039 DATA0,7,128,0,0,0,0,0
30041 DATA0,0,0,0,0,0,0
30042 DATA24,0,0,24,0,0,28
30043 DATA0,0,28,0,0,22,0
30044 DATA0,19,0,0,17,128,0
30045 DATA16,128,0,16,128,0,16
30046 DATA0,0,16,0,0,16,0
30047 DATA0,16,0,0,16,0,7
30048 DATA144,0,15,224,0,15,192
30049 DATA0,7,128,0,0,0,0,0
30051 DATA0,0,0,0,0,0,0
30052 DATA24,0,0,24,0,0,28
30053 DATA0,0,28,0,0,22,0
30054 DATA0,19,0,0,17,128,0
30055 DATA16,128,0,16,128,0,16
30056 DATA0,0,16,0,0,16,0
30057 DATA0,16,0,0,16,0,7
30058 DATA144,0,15,224,0,15,192
30059 DATA0,7,128,0,0,0,0,0
30061 DATA0,0,0,0,0,0,0
30062 DATA24,0,0,24,0,0,28
30063 DATA0,0,28,0,0,22,0
```

```
30064 DATA0,19,0,0,17,128,0
30065 DATA16,128,0,16,128,0,16
30066 DATA0,0,16,0,0,16,0
30067 DATA0,16,0,0,16,0,7
30068 DATA144,0,15,224,0,15,192
30069 DATA0,7,128,0,0,0,0,0
30070 DATA0,0,0,0,0,0,1
30071 DATA128,13,1,192,206,1
30072 DATA224,205,1,176,222,1,152
30073 DATA236,1,136,220,1,137,236
30074 DATA1,136,204,1,129,192,1
30075 DATA128,0
30076 DATA1,128,0,1,128,0
30077 DATA1,128,0,1,128,0,1
30078 DATA128,0,123,128,0,255,0
30079 DATA0,252,0,0,120,0,0,0
40000 FORI=15872TO16383:READA:POKEI,A:NEXT:V=53248
40001 FORI=0TO7:POKE2040+I,248+I:NEXT
40002 FORI=0TO7:POKEV+I,0:NEXT:POKEV+21,15:POKEV+2
3,0:POKEV+29,0
40004 RETURN
50000 REM MUSICAL AND SPRITE POSITIONS
50001 DIMNO(34),KE$(255),AN$(255),SP(34),NP(17),A(
17),B(17),C(17)
50002 DATA4,73,4,208,5,103,5,185,6,108,7,53,8,23,8
,147,9,159,10,205
50003 DATA4,139,5,25,6,16,6,206,7,163,9,21,10,60
50006 FORI=1TO34:READNO(I):NEXT
50008 KE$="ASDFGHJKL:WETYUOP"
50010 S=54272:POKES+3,10:POKES+2,255:POKES+5,9:POK
ES+6,0:POKES+4,65:POKE8+24,15
50016 FORI=1TO34:READSP(I):NEXT
50017 FORI=1TO17:READNP(I):NEXT
50018 RETURN
51000 DATA 41,190,65,190,89,190,113,190,137,190,16
1,190,185,190,209,190
51001 DATA233,190,255,190
51002 DATA 54,150,78,150,126,150,150,150,174,150,2
22,150,246,150
51007 DATA100,96,92,88,84,80,76,72,68,64,100,96,88
,84,80,72,68
```

### Explanation

Line 0 : Turn all sprites off.

Line 1 : Let's have a grey background and border, printing in upper and lower case rather than upper case and graphics, and finally print a short message on the screen.

Line 5 : Go to the routine at line 40000, which reads in the data for all the sprites. Then go to the routine starting at line 50000 and read in some more data, before clearing the screen, drawing up the keyboard and stave, and starting the program proper at line 110.

Lines 6-10 : Draw the stave up. Please note that the strange character at the start of lines 7, 8 and 9 is meant to represent a left hand square bracket and not an upper case letter L. A slight typing error there!

Line 11 : Expand sprites one and two in the X direction, locate sprite one on the screen and make it have a black colour.

Line 12 : Place sprite two on the screen, and make it also have a black colour.

Line 13 : The colour for sprite three is going to be orange, while all the rest of them will be white.

Line 18 : Print two cursor down characters.

Lines 19-105 : Draw the keyboard on the screen. Take great care over this, because the use of mnemonics has caused us to spill over on to two lines rather than just one, as the lines would be if drawn using the graphics characters.

Lines 110-130 : Simple loop to wait until the space bar is pressed before continuing.

Line 135 : Turn all sprites off.

Lines 140-170 : Program instructions.

Line 180 : Print four cursor up characters before going to the routine at line 18 to draw up the keyboard.

Line 182 : Turn some sprites on and give some colour to sprite three (useless really, since we've already done this!). Then go to the routine starting at line 185 to allow you to play some practice notes, and then go off to line 230.

Line 185 : Wait for a key to be pressed.

Line 186 : If that key is the exclamation mark then turn all sprites off and return from this routine.

Line 190 : Check to see whether the key pressed is equal to one of our control keys in the string KE$. If it is, then go to line 200.

Line 195 : Continue around the loop.

Line 196 : Back to wait for another key to be pressed.

Line 200 : Turn the waveform for voice one off then on again. This is always necessary on the 64.

Line 210 : POKE in the note value and the sprite position from the data read in via the subroutine at line 50000.

Line 220 : Back to the main body of this routine.

Line 230 : Clear the screen, print up a message, draw up the stave and keyboard before turning on each and every sprite.

Line 250 : Give all the musical note sprites an X co-ordinate.

Line 256 : Just a message.

Line 258 : Five note sprites, so we have a loop that will be executed five times.

Line 260 : Wait for a key to be pressed.

Line 261 : If that key is the exclamation mark, then turn all the sprites off, clear the screen and end the program.

Line 262 : Do a check for the ASCII value of the key pressed to prevent superfluous keys creeping through.

Line 264 : Set a variable equal to this ASCII value.

Line 268 : Start of a check on the key pressed to see if it matches one of our control keys.

Line 270 : Continue around the loop, and when finished go back for more.

Line 272 : Guess what?

Line 274 : Turn the waveform for voice one off then on again.

Line 276 : Put up the sprite position and play the note from the data read in via the routine starting at line 50000.

Line 277 : Couldn't fit this on to line 276.

Line 278 : If one of the flat notes has been played, then set the appropriate sprite data pointer to point to the area of memory that contains the data for a flat note sprite.

Line 279 : Let's have an ordinary note sprite back again.

Line 280 : Next step around the J loop before starting again.

Lines 30001-30079 : Sprite data, read in by ...

Line 40000 : Read all sprite data in and set the variable V.

Line 40001 : Set all the sprite data pointers.

Line 40002 : Some default positions and conditions for the sprites.

Line 40004 : End of routine.

Line 50000 : Simple enough.

Line 50001 : Declare a few arrays.

Lines 50002-50003 : Note values in high order, low order format.

Line 50006 : Read those note values in.

Line 50008 : Control keys that will actually produce a musical note.

Line 50010 : Set up some sound values.

Line 50016 : Read the values for the various note sprite positions on the keyboard.

Line 50017 : And ditto for positions on the stave.

Line 50018 : End of this routine.

Lines 51000-51002 : Sprite positions on the keyboard.

Line 51007 : And on the stave.

# Note recognition

Our second musical program for this chapter is a note recognition exercise. A note will be displayed on the stave, and you will have to play the correct note on the keyboard. The stave and keyboard used is virtually the same as in the previous program, so you could save your fingers some tapping by getting those lines in now.

```
0 POKE53248+21,0
1 POKE53281,5:POKE53280,1:POKE53272,23:PRINT"[CLR,
BLK]NOTE RECOGNITION."
5 GOSUB40000:GOSUB50000:PRINT"[HOME,2CD]":GOSUB6:G
OTO110
6 PRINT"[BLK,3SP,6SHIFT*,CBMR,23SHIFT*]
7 PRINT"[3SP,6SHIFT*,SHIFT+,23SHIFT*]
8 PRINT"[3SP,6SHIFT*,SHIFT+,23SHIFT*]
9 PRINT"[3SP,6SHIFT*,SHIFT+,23SHIFT*]
10 PRINT"[3SP,6SHIFT*,CBME,23SHIFT*]
11 POKEV+23,3:POKEV+1,70:POKEV,40:POKEV+39,0
12 POKEV+2,60:POKEV+3,70:POKEV+40,0
13 POKEV+41,8:FORI=42TO46:POKEV+I,1:NEXT
18 PRINT"[2CD]
19 PRINT"8SP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,RV
S,2SP,OFF,SP,RVS,2SP,OFF,SP,RVS]";
20 PRINT"[2SP,OFF,4SP,RVS,2SP,OFF]
21 REM LINES 19 AND 20 COMBINE TOGETHER TO FORM ON
E LINE ON THE 64
30 PRINT"8SP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,RV
S,2SP,OFF,SP,RVS,2SP,OFF,SP,RVS]";
31 PRINT"[2SP,OFF,4SP,RVS,2SP,OFF]
32 REM LINES 30 AND 31 COMBINE TOGETHER TO FORM ON
E LINE ON THE 64
40 PRINT"8SP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,RV
S,2SP,OFF,SP,RVS,2SP,OFF,SP,RVS]";
41 PRINT"[2SP,OFF,4SP,RVS,2SP,OFF]
42 REM LINES 40 AND 41 COMBINE TOGETHER TO FORM ON
E LINE ON THE 64
50 PRINT"[6SP,RVS,WHT,2SP,BLK,2SP,WHT,SP,BLK,2SP,W
HT,SP,CBMM,2SP,BLK,2SP,WHT]";
51 PRINT"[SP,BLK,2SO,WHT,BLK,2SP,WHT,SP,CBMM,2SP,B
LK,2SP]"
52 REM LINES 50 AND 51 COMBINE TOGETHER TO FORM ON
E LINE ON THE 64
60 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]";
61 PRINT"[2SP,CBMM,2SP,CBMM]"
```

```
62 REM LINES 60 AND 61 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
70 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]";
71 PRINT"[2SP,CBMM,2SP,CBMM]"
72 REM LINES 70 AND 71 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
80 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]";
81 PRINT"[2SP,CBMM,2SP,CBMM]"
82 REM LINES 80 AND 81 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
90 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM]";
91 PRINT"[2SP,CBMM,2SP,CBMM]"
92 REM LINES 90 AND 91 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
100 PRINT"[6SP.RVS.WHT.2SP,CBMM,2SP,CBMM,2SP,CBMM,
2SP,CBMM,2SP,CBMM,2SP,CBMM]";
101 PRINT"[2SP,CBMM,2SP,CBMM]"
102 REM LINES 100 AND 101 JOIN TOGETHER TO FORM ON
E LINE ON THE 64
105 RETURN
110 PRINT"[CD,BLK]PRESS 'SPACE' TO CONTINUE."
120 FORI=1TO10:GETSP$:NEXT
130 GETSP$:IFSP$<>" "THEN130
135 POKEV+21,0
140 PRINT"[CLR.BLK]IN THIS GAME I SHALL SHOW A NUM
BER OF   NOTES (NO BLACK NOTES: ";
150 PRINT"REFERENCE ONLY), AND YOU SHALL HAVE TO R
EPEAT THEM USING THE KEYBOARD.
160 PRINT"[CD]THE KEYBOARD IS PLAYED USING THE SAM
E   KEYS AS THE MUSICAL SIMON":
170 PRINT" GAME: TRY IT  NOW, USING KEYS A-K AND W
,E,T,Y,U,O :    '!' TO QUIT."
180 GOSUB18
182 POKEV+21,4:POKEV+41,8:GOSUB185:GOTO230
185 GETA$:IFA$=""THEN185
186 IFA$="!"THENRETURN
190 FORI=1TO14:IFA$=MID$(KE$,I,1)THEN200
195 NEXT
196 GOTO185
200 POKES+4,0:POKES+4,65
210 POKES+1,NO(I+I-1):POKES,NO(I+I):POKEV+4,SP(I+I
-1):POKEV+5,SP(I+I)
220 GOTO195
230 PRINT"[CLR.BLK]NOTE RECOGNITION.[2CD]":GOSUB6:
POKEV+21,255
240 CO=INT(RND(.5)*5)+1
250 FORI=1TOCO
252 A=INT(RND(.5)*8):A=72+A*4:POKEV+4+I*2.100+I*20
:POKEV+5+I*2,A
```

```
254 A(I)=A:B(I)=100+I*20:NEXT
256 PRINT"[2CD]NOW IT'S YOUR TURN : REPEAT THE NOT
ES!"
257 FORK=1TO10:GETA$:NEXT
258 FORJ=1TOCO
260 GETA$:IFA$=""THEN260
262 IFASC(A$)<65ORASC(A$)>90THEN260
264 A=ASC(A$)
268 FORI=1TO14:IFA$=MID$(KE$.I,1)THEN274
270 NEXTI:GOTO260
272 REM PLAY NOTE
274 POKES+4,0:POKES+4,65
276 POKES+1,NO(I+I-1):POKES,NO(I+I):POKEV+4.SP(I+I
-1):POKEV+5,SP(I+I)
278 POKEV+5+J*2,NP(I):C(J)=NP(I)
280 NEXTJ
284 FORI=1TOCO:IFA(I)<>C(I)THENPRINT"[HOME]
[HOME,RVS,BLK]WRONG!":Z=Z+1:ZQ=ZQ+1
288 NEXTI:IFZQ>OTHEN299
290 PRINT"[HOME]                [HOME.RVS.WHT]RIGHT!"
299 FORI=1TO1500:NEXT:Y=Y+CO-ZQ
300 PRINT"[HOME]          [HOME]RIGHT":Y:"     WRONG"
:Z:ZQ=0
301 FORI=OTO15:POKEV+I,0:NEXT
302 FORI=1TO3000:NEXT:GOTO230
30001 DATA0,0,0,0,8,0,0
30002 DATA34,0,0,66,0,0,66
30003 DATA0,0,66,0,0,68,0
30004 DATA0,88,0,0,224,0,3
30005 DATA32,0,4,124,0,8,162
30006 DATA0,9,49,0,9,17,0
30007 DATA8,146,0,4,20,0,3
30008 DATA248,0,0,16,0,12,16
30009 DATA0,12,48,0,7,192,0,0
30011 DATA0,0,0,0,0,0,0
30012 DATA0,0,0,0,0,0,0
30013 DATA0,0,0,0,0,24,0
30014 DATA0,126,0,0,153,0,1
30015 DATA30,0,1,24,0,1,24
30016 DATA0,1,24,0,0,154,0
30017 DATA0,124,0,0,24,0,0
30018 DATA0,0,0,0,0,0,0
30019 DATA0,0,0,0,0,0,0,0
30021 DATA0,0,0,0,0,0,0
30022 DATA24,0,0,24,0,0,28
30023 DATA0,0,28,0,0,22,0
30024 DATA0,19,0,0,17,128,0
30025 DATA16,128,0,16,128,0,16
30026 DATA0,0,16,0,0,16,0
30027 DATA0,16,0,0,16,0,7
30028 DATA144,0,15,224,0,15,192
30029 DATA0,7,128,0,0,0,0,0
```

```
30031 DATA0,0,0,0,0,0,0
30032 DATA24,0,0,24,0,0,28
30033 DATA0,0,28,0,0,22,0
30034 DATA0,19,0,0,17,128,0
30035 DATA16,128,0,16,128,0,16
30036 DATA0,0,16,0,0,16,0
30037 DATA0,16,0,0,16,0,7
30038 DATA144,0,15,224,0,15,192
30039 DATA0,7,128,0,0,0,0,0
30041 DATA0,0,0,0,0,0,0
30042 DATA24,0,0,24,0,0,28
30043 DATA0,0,28,0,0,22,0
30044 DATA0,19,0,0,17,128,0
30045 DATA16,128,0,16,128,0,16
30046 DATA0,0,16,0,0,16,0
30047 DATA0,16,0,0,16,0,7
30048 DATA144,0,15,224,0,15,192
30049 DATA0,7,128,0,0,0,0,0
30051 DATA0,0,0,0,0,0,0
30052 DATA24,0,0,24,0,0,28
30053 DATA0,0,28,0,0,22,0
30054 DATA0,19,0,0,17,128,0
30055 DATA16,128,0,16,128,0,16
30056 DATA0,0,16,0,0,16,0
30057 DATA0,16,0,0,16,0,7
30058 DATA144,0,15,224,0,15,192
30059 DATA0,7,128,0,0,0,0,0
30061 DATA0,0,0,0,0,0,0
30062 DATA24,0,0,24,0,0,28
30063 DATA0,0,28,0,0,22,0
30064 DATA0,19,0,0,17,128,0
30065 DATA16,128,0,16,128,0,16
30066 DATA0,0,16,0,0,16,0
30067 DATA0,16,0,0,16,0,7
30068 DATA144,0,15,224,0,15,192
30069 DATA0,7,128,0,0,0,0,0
30071 DATA0,0,0,0,0,0,0
30072 DATA24,0,0,24,0,0,28
30073 DATA0,0,28,0,0,22,0
30074 DATA0,19,0,0,17,128,0
30075 DATA16,128,0,16,128,0,16
30076 DATA0,0,16,0,0,16,0
30077 DATA0,16,0,0,16,0,7
30078 DATA144,0,15,224,0,15,192
30079 DATA0,7,128,0,0,0,0,0
40000 FORI=15872TO16383:READA:POKEI,A:NEXT:V=53248
40001 FORI=0TO7:POKE2040+I,248+I:NEXT
40002 FORI=0TO7:POKEV+I,0:NEXT:POKEV+21,15:POKEV+2
3,0:POKEV+29,0
40004 RETURN
50000 REM MUSICAL AND SPRITE POSITIONS
50001 DIMNO(28),KE$(255),AN$(255),SP(28),NP(14),A(
```

```
14),B(14),C(14)
50002 DATA4,73,4,208,5,103,5,185,6,108,7,53,8,23,8
,147,4,139,5,25,6,16,6,206
50004 DATA7,163,9,21
50006 FORI=1TO28:READNO(I):NEXT
50008 KE$="ASDFGHJKWETYUO"
50010 S=54272:POKES+3,10:POKES+2,255:POKES+5,9:POK
ES+6,0:POKES+4,65:POKES+24,15
50016 FORI=1TO28:READSP(I):NEXT
50017 FORI=1TO8:READNP(I):NEXT
50018 RETURN
51000 DATA 73,190,97,190,121,190,145,190,169,190,1
93,190,217,190,241,190
51002 DATA 86,150,110,150,158,150,182,150,206,150,
255,150
51004 DATA100,96,92,88,84,80,76,72
```

**Explanation**

Line 0 : Turn all sprites off.

Line 1 : Let's have a grey background and border, printing in upper and lower case rather than upper case and graphics, and finally print a short message on the screen.

Line 5 : Go to the routine at line 40000, which reads in the data for all the sprites. Then go to the routine starting at line 50000 and read in some more data, before clearing the screen, drawing up the keyboard and stave, and starting the program proper at line 110.

Lines 6-10 : Draw the stave up. Please note that the strange character at the start of lines 7, 8 and 9 is meant to represent a left hand square bracket and not an upper case letter L. A slight typing error there!

Line 11 : Expand sprites one and two in the X direction, locate sprite one on the screen and make it have a black colour.

Line 12 : Place sprite two on the screen, and make it also have a black colour.

Line 13 : The colour for sprite three is going to be orange, while all the rest of them will be white.

Line 18 : Print two cursor down characters.

Lines 19-105 : Draw the keyboard on the screen. Take great care over

this, because the use of mnemonics has caused us to spill over on to two lines rather than just one, as the lines would be if drawn using the graphics characters.

Lines 110-130 : Simple loop to wait until the space bar is pressed before continuing.

Line 135 : Turn all sprites off.

Lines 140-170 : Program instructions.

Line 180 : Off to line 18 to draw up the keyboard.

Line 182 : Turn some sprites on and give some colour to sprite three. Then go to the routine starting at line 185 to allow you to play some practice notes, and then go off to line 230.

Line 185 : Wait for a key to be pressed.

Line 186 : If that key is the exclamation mark then return from this routine.

Line 190 : Check to see whether the key pressed is equal to one of our control keys in the string KE$. If it is, then go to line 200.

Line 195 : Continue around the loop.

Line 196 : Back to wait for another key to be pressed.

Line 200 : Turn the waveform for voice one off then on again.

Line 210 : POKE in the note value and the sprite position from the data read in via the subroutine at line 50000.

Line 220 : Back to the main body of this routine.

Line 230 : Clear the screen, print up a message, draw up the stave and keyboard before turning on each and every sprite.

Line 240 : Pick a number, any number, between one and five.

Line 250 : For I equals 1 to that number.

Line 252 : Generate a random number between nought and seven. Turn that number into a vertical sprite position, and POKE that sprite

onto the screen.

Line 254 : Just store the values, for checking later.

Line 256 : Okay, it's the turn of the player to have a go now.

Line 257 : Clear the keyboard buffer so that an incorrect note isn't accidentally played.

Line 258 : Perform a loop CO times.

Line 260 : Wait for a key to be pressed.

Line 262 : If it isn't alphabetic then ignore it.

Line 264 : Take the ASCII value of the key pressed.

Line 268 : Check to see if it's a valid key.

Line 270 : Continue our check and then back for another note.

Line 272 : Bit obvious, really.

Line 274 : Turn the waveform for voice one off and on again.

Line 276 : Play note and display sprite on screen using data read in from routine starting at line 50000.

Line 278 : Rest of that play and display, and store the note played in the array C.

Line 280 : Get another note.

Line 284 : Compare the randomly generated notes with the played notes, and if the player's got one wrong, then tell him.

Line 288 : Continue the check, and if the player's got at least one note wrong then jump to line 299.

Line 290 : A miracle! Every note repeated correctly, so print out a message of congratulations.

Line 299 : Delay for a little while, before working out how many notes the player has got right overall.

Line 300 : Update the 'right/wrong' score, and reset the ZQ variable to zero.

Line 301 : Remove all the sprites.

Line 302 : Hang on for a while, before going back for more.

Lines 30001-30079 : Sprite data, read in by ...

Line 40000 : Read all sprite data in and set the variable V.

Line 40001 : Set all the sprite data pointers.

Line 40002 : Some default positions and conditions for the sprites.

Line 40004 : End of routine.

Line 50000 : Simple enough.

Line 50001 : Declare a few arrays.

Lines 50002-50003 : Note values in high order, low order format.

Line 50006 : Read those note values in.

Line 50008 : Control keys that will actually produce a musical note.

Line 50010 : Set up some sound values.

Line 50016 : Read the values for the various note sprite positions on the keyboard.

Line 50017 : And ditto for positions on the stave.

Line 50018 : End of this routine.

Lines 51000-51002 : Sprite positions on the keyboard.

Line 51007 : And on the stave.

Let's take a look at some filtering techniques now.

# 9
# Filtering Techniques

The 64 is probably one of the most powerful home computers when it comes to sound, and we've already seen how envelope shaping, ring modulation and synchronisation are all perfectly feasible. However, there is one other technique which we have yet to look at, and that involves filtering notes. There are a number of possibilities that we can use, and the synthesiser program in the next chapter explores them all. For now, let's start with a few definitions and some theory.

## Filtering

We've already stated that there are three main types of filter on the 64, and these are the low pass, high pass and band pass, which let through various frequencies in the tones but dismiss others according to various parameters. We can combine a high pass and a low pass filter together to get what is termed a notch filter, and with these four factors in mind, we can also alter the resonance of the filter. A high resonance gives a very intense effect, while a low resonance is altogether smoother.

By setting the resonance to be whatever we want, we can then set up various types of filter and, by altering the cutoff frequency so that differing frequencies come through while a note is playing, just one note can produce some interesting noises.

And although we haven't got one of the features of some very expensive synthesisers, namely an envelope generator that controls the filter cutoff so that it can automatically rise and fall while the note is playing, we can always get round this with a suitable piece of software.

Finally, each voice can either go through the filter or bypass it completely and go straight to the audio output.

The registers to look for are numbers 21, 22 and 23, as these are the

ones that control the cutoff frequency (21 and 22) and the resonance (23).

If bit 0 of register 23 is set to a 1 (i.e. you POKE V+23, PEEK(V+23)OR1) then voice 1 will be routed through the filter. POKEing it with OR2 will set voice 2 through the filter, and POKEing it with OR3 will put voice 3 through the filter.

On a scale of 1 to 15, the resonance is then selected by deciding what sort of resonance you want (15 being the maximum), and then POKEing the register with your resonance value times 16, plus of course the value for whatever voice, if any, you want to route through.

So, if you wanted maximum resonance and only voice 1 to go through the filter, you would have to POKE V+23 with 15 times 16 plus 1, or 241.

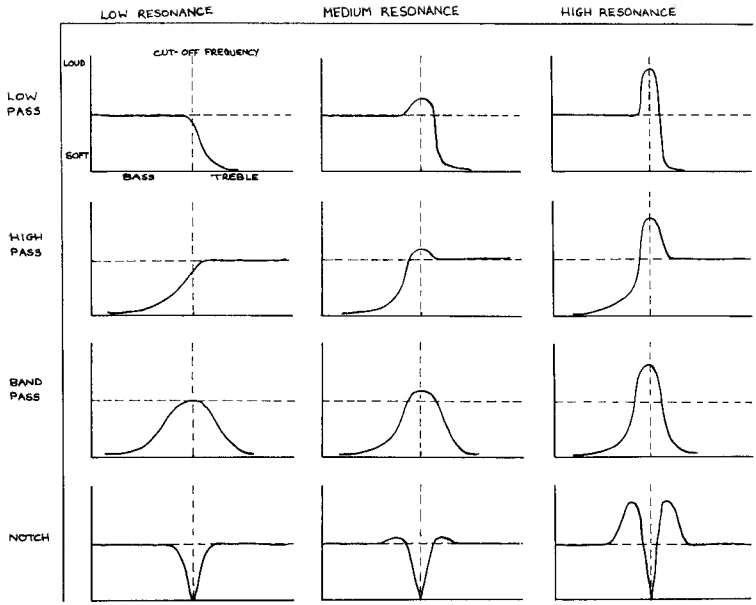
The type of filter that is used depends on the content of register V+24. Normally set to 15 to control the volume, the top four bits of this register can be altered to produce one of the four filter types.

Setting bit 4 (POKE it with 15 plus 16) selects low pass, bit 5 (POKE it with 15 plus 32) band pass, and bit 6 (POKE it with 15 plus 64) high pass. POKEing the register with 15 plus 128, i.e. setting bit 7, stops voice 3 from reaching any audio output, although it is still actively working away.

If we stop it going through the filter as well, voice 3 does not produce any sound at all, but it can still be used to modulate other voices without producing any odd noises of its own in the background.

Finally, to select the cutoff frequency, looking at the synthesiser program should help here.

To illustrate graphically what's going on, here are a few illustrations which point out just some of the effects that SID can achieve.




By careful adjustment of any or all of the parameters, the power of SID will slowly begin to become apparent to you. Even if you only produce noise at first, a noise is better than total silence. Well, usually.

As we said right at the start of this chapter, the synthesiser program in Chapter 10 explores most of the concepts explored here, so to get the most out of filtering it would be advisable to type that program in and get it up and running.

To end, a non-musical diversion. This program has nothing whatsoever to do with music. To prevent you from suffering a total aural assault throughout this entire book, here is a program that completely re-defines the Commodore 64 character set.

## Charset

Most of this program is data, apart from a brief setting up and explanatory set of notes at the start, so get those fingers busy!

```
5 POKE53272,23:GOSUB10000
10 POKE56333,127
20 POKE1,51
60 FORX=0TO1752
70 READA:POKE53248+X,A
80 NEXT
90 POKE1,55
100 POKE56333,129
110 POKE648,196
120 POKE56576,4
130 POKE53272,21
140 PRINT"[CLR,BLK,RVS]THE WORLD OF GRAPHICS!"
150 PRINT"[2CD]LIKE THIS :"
152 PRINT:PRINT:PRINT"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
154 PRINT"[CD,RVS,RED]ABCDEFGHIJKLMNOPQRSTUVWXYZ"
156 PRINT"[CD,BLU]ABCDEFGHIJKLMNOPQRSTUVWXYZ"
158 PRINT"[CD,RVS,BRO]ABCDEFGHIJKLMNOPQRSTUVWXYZ"
160 END
10000 POKE53280,9:POKE53281,7:PRINT"[CLR,RVS,BLK]W
ELCOME TO THE WORLD OF GRAPHICS!"
10001 PRINT
10002 PRINT"THIS PROGRAM ISN'T EXACTLY MUSICAL, BU
T YOU MIGHT FIND THE CHARACTER";
10004 PRINT" SET IT      PRODUCES USEFUL. IT COULD
BE USED IN AN ADVENTURE PROGRAM";
10006 PRINT" PERHAPS, OR INDEED ANYPROGRAM THAT RE
LIES HEAVILY ON TEXT."
10008 PRINT"[2CD]YOU'LL NOW HAVE TO WAIT A BIT WHI
LE THE NEW CHARACTER SET DATA ";
10010 PRINT"IS READ IN.  WHENWE'RE FINISHED, YOU'L
L BE ABLE TO TYPE  IN ";
10012 PRINT"UPPER AND LOWER CASE, REVERSE OR NON-R
EVERSE, IN ANY COLOUR YOU ";
10014 PRINT"FEEL LIKE,   AND THE CHARACTERS WILL A
LL APPEAR IN   THIS NEW FORMAT."
10016 PRINT"[2CD]HANG ON."
10018 RETURN
30001 DATA32,60,102,110,110,96,98,60
30002 DATA0,0,1,63,102,102,102,63
30003 DATA0,192,96,124,102,102,102,60
30004 DATA0,0,0,62,98,96,98,62
30005 DATA0,14,12,60,102,102,102,60
30006 DATA0,0,0,60,102,124,96,60
30007 DATA0,56,108,96,252,96,96,192
30008 DATA0,3,62,102,62,6,102,60:REMB
30009 DATA224,96,96,124,102,102,102,103
30011 DATA0,0,0,126,24,24,24,126
30012 DATA0,0,0,63,12,12,108,60
30013 DATA224,96,96,124,102,124,108,102:REMK
30014 DATA0,48,24,24,24,24,24,12
30015 DATA0,0,0,54,127,107,107,227
30016 DATA0,0,0,238,102,118,110,230
```

```
30017 DATA0,0,0,60,102,102,102,60
30019 DATA0,0,0,124,102,252,96,96:REMP
30020 DATA0,0,3,62,102,62,6,7
30021 DATA0,0,0,110,58,48,48,112
30022 DATA0,0,0,30,48,28,6,60
30023 DATA24,24,24,254,152,24,27,31:REMT
30024 DATA0,0,0,238,102,102,102,63
30025 DATA0,0,0,102,102,102,60,24
30026 DATA0,0,0,99,107,127,119,227
30028 DATA0,0,0,102,60,24,60,102
30029 DATA0,3,102,102,62,6,102,60
30030 DATA0,0,0,126,12,48,98,126:REMZ
30031 DATA0,60,48,48,48,48,48,60
30032 DATA0,12,18,48,124,48,98,252
30033 DATA0,60,12,12,12,12,12,60
30034 DATA0,0,24,60,126,24,24,24
30035 DATA24,0,16,48,127,127,48,16
30037 DATA0,0,0,0,0,0,0,0
30038 DATA0,24,24,24,24,0,0,24
30039 DATA0,102,102,102,0,0,0,0
30040 DATA0,102,102,255,102,255,102,102
30041 DATA0,24,62,96,60,6,124,24
30042 DATA0,98,102,12,24,48,102,70
30043 DATA0,60,102,60,56,103,102,63
30044 DATA0,6,12,24,0,0,0,0
30046 DATA0,12,24,48,48,48,24,12
30047 DATA0,48,24,12,12,12,24,48
30048 DATA0,0,102,60,255,60,102,0
30049 DATA0,0,24,24,126,24,24,0
30050 DATA0,0,0,0,0,24,24,48
30051 DATA0,0,0,0,126,0,0,0
30052 DATA0,0,0,0,0,0,24,24
30053 DATA0,0,3,6,12,24,48,96
30055 DATA0,60,102,110,118,102,102,60
30056 DATA0,24,24,56,24,24,24,126
30057 DATA0,60,102,6,12,48,96,126
30058 DATA0,60,102,6,28,6,102,60
30059 DATA0,6,14,30,102,127,6,6
30060 DATA0,126,96,124,6,6,102,60
30061 DATA0,60,102,96,124,102,102,60
30062 DATA0,126,102,12,24,24,24,24:REM7
30064 DATA0,60,102,102,60,102,102,60
30065 DATA0,60,102,102,62,6,102,60
30066 DATA0,0,0,24,0,0,24,0
30067 DATA0,0,0,24,0,0,24,24:REM SEMI COLON
30068 DATA0,14,24,48,96,48,24,14
30069 DATA0,0,0,126,0,126,0,0
30070 DATA0,112,24,12,6,12,24,112
30071 DATA0,60,102,6,12,24,0,24
30073 DATA32,0,0,0,255,255,0,0
30074 DATA0,28,54,102,102,102,102,63:REMA
30075 DATA0,248,108,108,60,102,102,252
```

```
30076 DATA0,60,102,192,192,194,102,60
30077 DATA0,112,120,108,102,102,110,120
30078 DATA0,248,110,96,120,96,110,248
30079 DATA128,248,110,96,252,100,96,224
30080 DATA0,60,102,192,220,198,102,60
30082 DATA0,239,102,102,62,102,102,231
30083 DATA0,124,24,24,24,24,24,124:REMI
30084 DATA0,62,28,12,12,76,108,56
30085 DATA0,239,102,108,56,108,102,231
30086 DATA192,224,96,96,96,96,110,248
30087 DATA192,227,119,127,107,99,99,227
30088 DATA192,230,118,118,126,110,110,230:REMN
30089 DATA0,60,102,195,195,195,102,60
30091 DATA0,252,102,102,124,96,96,224
30092 DATA192,124,102,102,102,102,60,14
30093 DATA0,252,102,102,124,108,102,230
30094 DATA0,60,102,112,28,198,102,60
30095 DATA0,254,154,24,24,24,24,56
30096 DATA192,227,102,102,102,102,102,63
30097 DATA3,231,102,102,102,102,60,24
30098 DATA0,227,99,99,107,127,119,227
30100 DATA128,227,102,60,24,60,102,199
30101 DATA0,231,102,102,60,24,24,56
30102 DATA224,126,6,12,24,48,102,126
30103 DATA0,0,48,124,220,222,155,219
30104 DATA24,12,18,48,124,48,98,252
30105 DATA0,24,24,24,24,24,24,24
30106 DATA24,0,0,66,255,255,66,0
30107 DATA0,60,126,60,60,60,60,126
30109 DATA0,0,0,0,0,0,0,0
30110 DATA0,237,222,175,118,137,221,190
30111 DATA190,102,102,102,0,0,0,0
30112 DATA0,102,102,255,102,255,102,102
30113 DATA0,24,62,96,60,6,124,24
30114 DATA0,98,102,12,24,48,102,70
30115 DATA0,0,60,126,195,255,255,255
30116 DATA0,6,12,24,0,0,0,0
30118 DATA0,60,102,60,153,102,56,44
30119 DATA195,48,24,12,12,12,24,48
30120 DATA0,0,102,60,255,60,102,0
30121 DATA0,0,24,24,126,24,24,0
30122 DATA0,0,0,0,0,0,24,24
30123 DATA48,24,24,24,31,31,0,0
30124 DATA0,0,0,0,248,248,24,24
30125 DATA24,0,3,6,12,24,48,96
30127 DATA0,0,0,0,31,31,24,24
30128 DATA24,24,24,56,24,24,24,126
30129 DATA0,60,102,6,12,48,96,126
30130 DATA0,60,102,6,28,6,102,60
30131 DATA0,6,14,30,102,127,6,6
30132 DATA0,126,96,124,6,6,102,60
30133 DATA0,60,102,96,124,102,102,60
```

```
30134 DATA0,126,102,12,24,24,24,24
30136 DATA0,60,102,102,60,102,102,60
30137 DATA0,60,102,102,62,6,102,60
30138 DATA0,0,0,24,0,0,24,0
30139 DATA0,0,0,24,0,0,24,24
30140 DATA48,14,24,48,96,48,24,14
30141 DATA0,24,24,24,248,248,0,0
30142 DATA0,112,24,12,6,12,24,112
30143 DATA0,60,102,6,12,24,0,255
30145 DATA223,195,153,145,145,159,157,195
30146 DATA255,255,254,192,153,153,153,192
30147 DATA255,63,159,131,153,153,153,195
30148 DATA255,255,255,193,157,159,157,193
30149 DATA255,241,243,195,153,153,153,195
30150 DATA255,255,255,195,153,131,159,195
30151 DATA255,199,147,159,3,159,159,63
30152 DATA255,252,193,153,193,249,153,195:REMG
30154 DATA31,159,159,131,153,153,153,152
30155 DATA255,255,255,129,231,231,231,129
30156 DATA255,255,255,193,243,243,147,195
30157 DATA31,159,159,131,153,131,147,153:REMK
30158 DATA255,207,231,231,231,231,231,243
30159 DATA255,255,255,201,128,148,148,28
30160 DATA255,255,255,17,153,137,145,25
30161 DATA255,255,255,195,153,153,153,195
30163 DATA255,255,255,131,153,3,159,159:REMP
30164 DATA255,255,252,193,153,193,249,248
30165 DATA255,255,255,145,197,207,207,143
30166 DATA255,255,255,227,207,227,249,195
30167 DATA231,231,231,1,103,231,228,224:REMT
30168 DATA255,255,255,17,153,153,153,192
30169 DATA255,255,255,153,153,153,195,231
30170 DATA255,255,255,156,148,128,136,28
30172 DATA255,255,255,153,193,231,193,153
30173 DATA255,252,153,153,193,249,153,195
30174 DATA255,255,255,129,243,207,157,129:REMZ
30175 DATA255,195,207,207,207,207,207,195
30176 DATA255,243,237,207,131,207,157,3
30177 DATA255,195,243,243,243,243,243,195
30178 DATA255,255,231,195,129,231,231,231
30179 DATA231,255,239,207,128,128,207,239
30181 DATA255,255,255,255,255,255,255,255
30182 DATA255,231,231,231,231,255,255,231
30183 DATA255,153,153,153,255,255,255,255
30184 DATA255,153,153,0,153,0,153,153
30185 DATA255,231,193,159,195,249,131,231
30186 DATA255,157,153,243,231,207,153,185
30187 DATA255,195,153,195,199,152,153,192
30188 DATA255,249,243,231,255,255,255,255
30190 DATA255,243,231,207,207,207,231,243
30191 DATA255,207,231,243,243,243,231,207
30192 DATA255,255,153,195,0,195,153,255
```

```
30193 DATA255,255,231,231,129,231,231,255:REM PLUS
30194 DATA255,255,255,255,255,231,231,207
30195 DATA255,255,255,255,129,255,255,255
30196 DATA255,255,255,255,255,255,231,231
30197 DATA255,255,252,249,243,231,207,159
30199 DATA255,195,153,145,137,153,153,195
30200 DATA255,231,231,199,231,231,231,129
30201 DATA255,195,153,249,243,207,159,129
30202 DATA255,195,153,249,227,249,153,195
30203 DATA255,249,241,225,153,128,249,249
30204 DATA255,129,159,131,249,249,153,195
30205 DATA255,195,153,159,131,153,153,195
30206 DATA255,129,153,243,231,231,231,231
30208 DATA255,195,153,153,195,153,153,195
30209 DATA255,195,153,153,193,249,153,195
30210 DATA255,255,255,231,255,255,231,255
30211 DATA255,255,255,231,255,255,231,231
30212 DATA255,241,231,207,159,207,231,241
30213 DATA255,255,255,129,255,129,255,255
30214 DATA255,143,231,243,249,243,231,143
30215 DATA255,195,253,249,243,231,255,231
30216 DATA223,255,255,255,0,0,255,255
30217 DATA255,227,201,153,153,153,153,192
30218 DATA255,7,147,147,195,153,153,3
30219 DATA255,195,153,63,63,61,153,195
30220 DATA255,143,135,147,153,153,145,135
30221 DATA255,7,145,159,135,159,145,7
30222 DATA255,7,145,159,3,155,159,31
30223 DATA255,195,153,63,35,57,153,195
30224 DATA255,16,153,153,129,153,153,24
30225 DATA255,131,231,231,231,231,231,131:REMI
30226 DATA255,193,227,243,243,179,147,199
30227 DATA255,16,153,147,199,147,153,24
30228 DATA255,31,159,159,159,159,145,7
30229 DATA255,28,136,128,148,156,156,28
30230 DATA255,25,137,137,129,145,145,25:REMN
30231 DATA255,195,153,60,60,60,153,195
30232 DATA255,3,153,153,131,159,159,31
30233 DATA255,195,153,153,153,153,195,241
30234 DATA255,3,153,153,131,147,153,25
30235 DATA255,195,153,143,227,57,153,195
30236 DATA255,1,101,231,231,231,231,199
30237 DATA255,28,153,153,153,153,153,192
30238 DATA255,24,153,153,153,153,195,231
30239 DATA255,28,156,156,148,128,136,28
30240 DATA255,28,153,195,131,195,153,56
30241 DATA255,24,153,153,195,231,231,199
30242 DATA255,193,153,243,231,207,189,129
30243 DATA255,255,255,255,255,255,255,255
30244 DATA255,255,255,255,255,255,255,255
30245 DATA255,255,255,255,255,255,255,255
30246 DATA255,255,255,255,255,255,255,255
30247 DATA255,255,255,255,255,255,255,255
```

## Explanation

Line 5 : Go into upper case/lower case mode, and off to the routine at 10000 to print up some on-screen instructions.

Line 10 : disable interrupts while we change everything.

Line 20 : switch out input/output ROM.

Line 60 : read in character data.

Line 70 : poke it all into place.

Line 80 : next step around the loop.

Line 90 : replace input/output ROM.

Line 100 : re-enable interrupts.

Lines 110-130 : set pointers for video chip to access new character base and video RAM location.

Lines 140-160 : just a demonstration.

Lines 10000-10018 : and some instructions.

Lines 30001-30247 : the data!

# 10
# A Synthesiser

We've already seen how sophisticated the 64 can be, so it's about time we put some of that sophistication to the test.

This program uses all three voices, with voices two and three providing background rhythms using a method to be outlined in Chapter 13. What may be of interest is how those background rhythms can be used to simulate various musical instruments, and some possible settings for attack/decay and sustain/release are given below in order for you to be able to do precisely that.

**Possible settings for musical instruments**

| Instrument | Att/Dec | Sus/Rel | Waveform | Pulse |
|---|---|---|---|---|
| Piano | 9 | 0 | Pulse | Lo=255 |
| Harpsichord | 9 | 0 | Sawtooth | - |
| Trumpet | 96 | 0 | Sawtooth | - |
| Flute | 96 | 0 | Triangle | - |
| Xylophone | 9 | 0 | Triangle | - |
| Organ | 9 | 0 | Triangle | - |
| Accordion | 102 | 240 | Triangle | - |

## The program

This is a variation on a popular program, which goes a lot further than anything we've attempted to do so far. This program gives you most

of the power of a full synthesiser, including the capability to define any voice to 'perform' as you would like it to; set up and change background rhythms; have glissando effects; change the filtering parameters, and so on. In short, just about everything is included here.

```
1 PRINT"[CLR]":IFPEEK(49219)<>65THENLOAD "BOOGIETI
ME",8,1
2 GOSUB4000:PRINTCHR$(8)
5 V=54272:VO(0)=1:OC=4
15 GOSUB595
18 GOSUB800
20 GOSUB505
21 SYS 49152
25 GETKY$:IFKY$=""THEN25
26 K=PEEK(197):PS=PEEK(653)
27 IFK=57THENFORI=0TO24:POKEV+I,0:NEXT:PRINT"[CLR.
2CD]BYE.":END
28 IFK=48THEN5000
30 IFK=40THEN1999
31 IFK=4THEN6000
35 IFK=43THEN999
60 IF K=1 THEN250
65 IFK=46THENOC=OC/2:IFOC<1THENOC=1
67 IFK=35THENPOKEV+24,0:GOTO25
70 IFK=54THENOC=OC*2:IFOC>64THENOC=64
75 IFK=49THENGL=1-GL
80 IFK=53THENGR=GR+1:IFGR>8THENGR=0
90 F=N(K):LK=K:LS=PS
95 IF F=0 THEN 25
100 IF (F>0ANDF<9)THEN 225
105 F=F*(4/OC)
110 IFGLANDGR>0ANDZ<>FTHEN455
120 IF PS=1 THEN F=INT(F*2^(1/12))
130 F1=INT(F/256)
135 F2=F-F1*256
136 POKEV+24,15
140 REM
150 POKE V+4,0
155 POKE V+4,W(0)*16+RM(0)*2+SY(0)*4+1
165 POKE V,F2
170 IFF1>255THEN185
175 POKE V+1,F1
185 Z=F
190 GOTO 25
195 FOR I=0 TO 2
200 POKE V+I*7,0
205 POKE V+I*7+1,0
210 POKE V+I*7+4,W(I)*16
```

```
215 NEXT I
220 GOTO 25
225 F=F-1
230 FOR I=0 TO 2
235 VO(I)=(FAND2^I)/2^I
240 NEXT I
241 IFF=1THENPOKE49219,0:POKE51267,0:GOTO245
242 IFF=1ORF=2THENPOKE49219,0:POKE51267,W(1)*16+1+
RM(1)*4+SY(1)*2:GOTO245
243 IFF=0ORF=1ORF=3ORF=4THENPOKE51267,0:POKE49219,
W(2)*16+1+RM(2)*4+SY(2)*2:GOTO245
244 POKE49219,W(2)*16+1+RM(2)*4+SY(2)*2:POKE51267,
W(1)*16+1+RM(1)*4+SY(1)*2
245 GOTO 25
250 POKE53280,9:POKE53281,7
255 PRINT "[CLR,BLK]              VOICE 1   VOICE 2
  VOICE 3"
260 FORI=1TO10:GETKY$:NEXT
265 PRINT "[CD]WAVEFORM":TAB(12):W$(0):TAB(22):W$(
1):TAB(32):W$(2)
270 PRINT "ATT/DEC":TAB(13):AD(0):TAB(23):AD(1):TA
B(32):AD(2)
275 PRINT "SUS/REL":TAB(13):SR(0):TAB(23):SR(1):TA
B(32):SR(2)
280 PRINT "PULSE HI":TAB(13):PH(0):TAB(23):PH(1):T
AB(32):PH(2)
285 PRINT "PULSE LO":TAB(13):PL(0):TAB(23):PL(1):T
AB(32):PL(2)
290 PRINT "RING MOD":TAB(13):RM(0):TAB(23):RM(1):T
AB(32):RM(2)
295 PRINT "SYNC     ":TAB(13):SY(0):TAB(23):SY(1):T
AB(32):SY(2)
300 PRINT"[CD]DO YOU WANT TO CHANGE ANY VALUES (Y/
N)?"
305 GETCH$:IFCH$="N"THENPOKE53280,0:POKE53281,0:GO
TO20
310 IFCH$<>"Y"THEN305
315 PRINT"[CD]WHICH VOICE (1, 2 OR 3)?"
320 GETVC$:IFVC$=""THEN320
325 IFVC$="1"THENPRINT"VOICE 1":VC=0:GOTO 345
330 IFVC$="2"THENPRINT"VOICE 2":VC=1:GOTO 345
335 IFVC$="3"THENPRINT"VOICE 3":VC=2:GOTO 345
340 GOTO 320
345 PRINT "[CD]WAVEFORM (T, S, P, OR N)?"
350 GETWF$:IFWF$=""THEN 350
355 IFWF$="T"THENPRINT"TRIANGLE":W(VC)=1:W$(VC)="T
RIANGLE":GOTO 380
360 IFWF$="S"THENPRINT"SAWTOOTH":W(VC)=2:W$(VC)="S
AWTOOTH":GOTO 380
365 IFWF$="P"THENPRINT"PULSE":W(VC)=4:W$(VC)="PULS
E":GOTO 380
370 IFWF$="N"THENPRINT"NOISE":W(VC)=8:W$(VC)="NOIS
```

```
E":GOTO 380
375 GOTO 350
380 INPUT "ATTACK/DECAY":AD(VC):IFAD(VC)<00RAD(VC)
>255THENPRINT"[2CU]":GOTO 380
385 INPUT "SUSTAIN/RELEASE":SR(VC):IFSR(VC)<00RSR(
VC)>255THENPRINT"[2CU]":GOTO385
390 INPUT "PULSE HI":PH(VC):IFPH(VC)<00RPH(VC)>255
THENPRINT"[2CU]":GOTO 390
395 INPUT "PULSE LI":PL(VC):IFPL(VC)<00RPL(VC)>255
THENPRINT"[2CU]":GOTO 395
400 INPUT "RING MOD":RM(VC):IFRM(VC)<00RRM(VC)>1TH
ENPRINT"[2CU]":GOTO 400
405 INPUT "SYNC":SY(VC):IFSY(VC)<00RSY(VC)>1THENPR
INT"[2CU]":GOTO 405
406 IFVC=2THENPOKE49219,W(VC)*16+1+SY(VC)*2+RM(VC)
*4
407 IFVC=1THENPOKE51267,W(VC)*16+1+SY(VC)*2+RM(VC)
*4
410 GOTO 250
420 FORI=0TO2
425 IFVO(I)=0THEN435
430 POKEV+I*7+4,W(I)*16+2
435 NEXTI
440 IFPEEK(197)=64THEN420
445 GOTO25
450 W(0)=1:POKEV+4,W(0)*16+5:V(2)=1:GOTO25
455 IFZ>FTHENFR=-1:GOTO465
460 FR=1
465 FORI=ZTOFSTEPFR*GR*64
470 F1=INT(I/256)
475 F2=I-F1*256
480 IFRM=1THENPOKEV+4,W(0)*16+5
485 POKE V,F2
490 IFF1>255ORF1<0THEN500
495 POKE V+1,F1
500 NEXTI:Z=I:GOTO130
505 POKE 53280,0:POKE 53281,0:POKE 53272,23
510 PRINT"[YEL,CLR,RVS]*********    64 SYNTHESISER
    *********[OFF]"
515 PRINT"[RVS,CU]**********       BY SID       **
*********"
520 PRINT" PLAY USING THE KEYS [RVS] Q W E R T Y U
 I "
521 PRINT"                         [RVS]
   "
525 PRINT"                         [RVS] A S D F G H J
 K "
526 PRINT"                         [RVS]
   "
530 PRINT"                         [RVS] Z X C V B N M
 , "
531 PRINT"                         [RVS]
```

```
   [CU]":IFPP=1THENPP=0:RETURN
532 ZZ=1:IFZZ=1THENGOSUB805
585 GOSUB690
590 RETURN
595 DIM N(64)
600 FOR I=0 TO 64
605 READ A
610 N(I)=A
615 NEXT I
620 DATA .-1,0,0,0,0.0.0
625 DATA 4,9854,4389.5,2195,4927
630 DATA 11060.0.6,11718.5530.7.2765,5859
635 DATA 13153,2463,8,14764,6577,0,3288.7382
640 DATA 16572,2930.0,17557,8286,1,4143,8779
645 DATA 0,3691,0,0,0,0,0,0,0,4389,0.0.0
650 DATA 0,0,0,0,0,2,0,0,3,0
655 DATA 0,8779,0,0
660 FOR I=0 TO 2
665 READ W(I).AD(I),SR(I).PH(I).PL(I).W$(I).RM(I).
SY(I)
670 NEXT
675 DATA 2,102,108,0.0,"SAWTOOTH",0.0
680 DATA 4,9,0,0,255,"PULSE",0,0
685 DATA 4.9,0,0,255,"PULSE".0,0
690 FORI=0TO2
695 POKE V+7*I+4,W(I)+RM(I)*2+SY(I)*4
700 POKE V+7*I+5,AD(I):POKE V+7*I+6.SR(I):POKE V+7
*I+3,PH(I):POKE V+7*I+2,PL(I)
705 NEXT
710 POKE V+24,15
712 POKE49219,W(2)*16+RM(2)*4+SY(2)*2+1
714 POKE51267,W(1)*16+RM(1)*4+SY(1)*2+1
715 RETURN
800 POKE53280,0:POKE53281,0:POKE53272,23:PRINT"[CL
R,YEL]WELCOME TO 64-SYNTH"
802 PRINT"[2CD]PLAY USING THE KEYBOARD AS SHOWN ON
 THE DIAGRAM COMING UP SOON."
804 PRINT"[CD]JUST A FEW THINGS TO REMEMBER."
805 PRINT"[CD]0) PRESS '\' TO START RHYTHMS TOGETH
ER."
806 PRINT"1) PRESS SHIFT FOR A SHARP."
808 PRINT"2) PRESS RETURN TO ALTER VOICES."
810 PRINT"3) PRESS '*' FOR GLISSANDO (VOICE1)
812 PRINT"4) PRESS '=' TO ALTER GLISSANDO RATE."
814 PRINT"5) PRESS '@' TO GO UP AN OCTAVE."
815 PRINT"6) PRESS '^' TO GO DOWN AN OCTAVE."
816 PRINT"7) PRESS ' ' TO EXIT."
817 PRINT"8) PRESS '0' TO CANCEL ALL VOICES."
818 PRINT"9) PRESS 0-7 TO SWITCH ON/OFF VOICES."
819 PRINT"10) PRESS '+' TO ALTER RHYTHM 1.
820 PRINT"11) PRESS '-' TO ALTER RHYTHM 2.":PRINT"
12) PRESS 'F1' FOR FILTER."
```

```
821 IFZZ=1THENZZ=0:RETURN
822 GOSUB870
825 PRINT"[CLR]VOICES ARE ALTERED ON A BINARY BASI
S.   THUS :"
826 PRINT"[CD]PRESSING 1 TURNS ON VOICE  1."
828 PRINT"[CD]PRESSING 2 TURNS ON VOICE  2."
830 PRINT"[CD]PRESSING 3 TURNS ON VOICES 1 & 2."
831 PRINT"[CD]PRESSING 4 TURNS ON VOICE  3."
832 PRINT"[CD]PRESSING 5 TURNS ON VOICES 1 & 3."
834 PRINT"[CD]PRESSING 6 TURNS ON VOICES 2 & 3."
836 PRINT"[CD]PRESSING 7 TURNS ON VOICES 1. 2 & 3.
"
838 GOSUB870
840 PRINT"[CLR] BEST EFFECTS ARE OBTAINED WHEN YOU
  ALTER THE RING MODULATION ":
842 PRINT"AND SYNCHRON- ISATION FOR EACH VOICE."
844 PRINT"[2CD]WHEN ALTERING THESE, JUST ENTER 0 O
R 1  TO TURN THESE FEATURES OFF ":
846 PRINT"OR ON FOR    EACH VOICE."
848 PRINT"[2CD]THE REST IS UP TO YOU!"
870 PRINT"[CD]PRESS SPACE WHEN READY."
872 GETCO$:IFCO$<>" "THEN872
874 RETURN
999 POKE49219,0:POKE51267,W(1)*16+RM(1)*4+SY(1)*2+
1:SYS49171
1000 PP=1:GOSUB505:D=0
1001 GOSUB3000
1005 GETA$:IFA$=""THEN1005
1006 K=PEEK(197):PS=PEEK(653)
1007 IFK=1ANDD>2THENGOSUB505:POKE51379+D,255:POKE5
1380+D,255:GOTO20
1008 IFK=1THEN1005
1009 IFK=60THENPOKE51379+D,00:POKE51380+D,00:GOTO1
122
1010 IFK=46THENOC=OC/2:IFOC<1THENOC=1
1011 IFK=0THEN20
1013 IFK=57THEND=D-2:IFD<0THEND=0
1015 IFK=4THENSYS49152:GOTO1006
1016 IFK=5THENSYS49171:GOTO1006
1017 IFK=6THEND=0:GOTO1006
1020 IFK=54THENOC=OC*2:IFOC>64THENOC=64
1030 F=N(K):LK=K:LS=PS
1035 IFF=0THEN1005
1040 F=F*(4/OC)
1050 IF PS=1 THEN F=INT(F*2^(1/12))
1060 F1=INT(F/256)
1070 F2=F-F1*256
1075 POKEV+11,0
1090 POKE V+7,F2:POKE51380+D,F2
1100 IFF1>255THEN1120
1110 POKE V+8,F1:POKE51379+D,F1
1111 POKEV+11,W(1)*16+1+RM(1)*4+SY(1)*2
```

```
1120 IFD>900THENK=1:GOTO1008
1122 POKE51382+D,255:POKE51381+D,255
1125 D=D+2:GOTO1005
1999 POKE51267,0:POKE49219,W(2)*16+RM(2)*4+SY(2)*2
+1:SYS49171
2000 PP=1:GOSUB505:D=0
2001 GOSUB3000
2005 GETA$:IFA$=""THEN2005
2006 K=PEEK(197):PS=PEEK(653)
2007 IFK=1ANDD>2THENPOKE49331+D,255:POKE49332+D,25
5:GOTO20
2008 IFK=1THEN2005
2009 IFK=60THENPOKE49331+D,00:POKE49332+D,00:GOTO2
122
2010 IFK=46THENOC=OC/2:IFOC<1THENOC=1
2011 IFK=0THEN20
2013 IFK=57THEND=D-2:IFD<0THEND=0
2015 IFK=4THENSYS49152:GOTO2006
2016 IFK=5THENSYS49171:GOTO2006
2017 IFK=6THEND=0:GOTO2006
2020 IFK=54THENOC=OC*2:IFOC>64THENOC=64
2030 F=N(K):LK=K:LS=PS
2035 IFF=0THEN2005
2040 F=F*(4/OC)
2050 IF PS=1 THEN F=INT(F*2^(1/12))
2060 F1=INT(F/256)
2070 F2=F-F1*256
2075 POKEV+18,0
2090 POKE V+14,F2:POKE49332+D,F2
2100 IFF1>255THEN2120
2110 POKE V+15,F1:POKE49331+D,F1
2111 POKEV+18,W(2)*16+1+RM(2)*4+SY(2)*2
2120 IFD>900THENK=1:GOTO2008
2122 POKE49334+D,255:POKE49333+D,255
2125 D=D+2:GOTO2005
3000 PRINT"[RVS,PUR,2CD]'SPACE'[OFF] TO INTRODUCE
A PAUSE.
3002 PRINT"[RVS]'RETURN'[OFF] TO RETURN TO MAIN ME
NU.
3003 PRINT"[RVS]'F1'[OFF] TO PLAY TUNE.":PRINT"[RV
S]'F3'[OFF] TO CANCEL IT."
3004 PRINT"[RVS]'@'[OFF] TO GO UP AN OCTAVE.
3005 PRINT"[RVS] '[OFF] TO GO BACK A NOTE.":PRINT"
[RVS]'F5'[OFF] TO GO BACK TO START."
3006 PRINT"[RVS]'^'[OFF] TO GO DOWN AN OCTAVE.
3007 PRINT"[RVS]'INST/DEL[OFF] TO QUIT."
3008 PRINT"[CD]A MAXIMUM OF 900 NOTES ALLOWED."
3010 RETURN
4000 S=54272:POKES+17,255:POKES+19,9:POKES+20,0
4010 POKES+24,15:POKE49216,10:POKE49218,0:POKE4921
9,65:POKE251,178:POKE252,192
4020 POKE253,178:POKE254,192
```

```
4030 POKES+10,255:POKES+12,9:POKES+13,0
4040 POKE51264,10:POKE51266,0:POKE51267,65:POKE247
,178:POKE248,200
4050 POKE249,178:POKE250,200:RETURN
5000 POKE251,178:POKE252,192:POKE253,178:POKE254,1
92
5002 POKE247,178:POKE248,200:POKE249,178:POKE250,2
00
5004 GOTO25
6000 POKE53281,7:POKE53280,9:PRINT"[CLR,BLK]FILTER
ING ALTERATION
6002 PRINT"[CD]'F1' - LOW VALUE OF FREQUENCY CUT-O
FF."
6004 PRINT"'F3' - HIGH VALUE OF FREQUENCY CUT-OFF.
"
6006 PRINT"'F5' - SIGNAL ROUTED THROUGH FILTER."
6008 PRINT"'F7' - RESONANCE OF FILTER."
6010 PRINT"'F2' - FILTER MODE."
6015 PRINT"'!'  - EXIT."
6020 PRINT"[2CD]LOW VALUE  HIGH VALUE FREQUENCY CU
T-OFF
6022 PRINT"[4CD]SIGNAL  RESONANCE  MODE"
6027 C=A*256+B:C=(30+C*5.8)
6028 PRINT"[HOME,11CD,3CL]";A;"          ";B;"
   ";C;"HZ
6030 PRINT"[4CD,CL]";AA;"       ";E;"       ";FF
6040 GETA$:IFA$=""THEN6040
6041 IFA$="[F1]"THENB=B+1:IFB>255THENB=0
6042 IFA$="[F3]"THENA=A+1:IFA>7THENA=0
6044 IFA$="[F5]"THENAA=AA+1:IFAA>15THENAA=0
6046 IFA$="[F7]"THENE=E+1:IFE>15THENE=0
6048 IFA$="[F2]"THENFF=FF+1:IFFF>15THENFF=0
6050 POKEV+22,A:POKEV+21,B:POKEV+23,AA+E*16:POKEV+
24,15+FF*16
6056 IFA$="!"THEN20
6058 GOTO6000
```

**Explanation**

Line 1 : If you haven't got a background rhythm in, load one in from disk (this boogie rhythm is to be found in Chapter 13). To load from tape, change the number eight to a one.

Line 2 : GOSUB 4000 to set up the background rhythm voice driver, and then disable the action of the shift and CBM logo keys.

Line 5 : declare variable V, and turn voice 1 on (all voices are referenced as VO), as well as setting the octave number.

Line 15 : go to subroutine to read all the musical data in.

Line 18 : go to the routine to print some on-screen instructions.

Line 20 : go to subroutine to print on-screen instructions and keyboard.

Line 21 : start the machine code interrupt-driven background voices going.

Line 25 : check for key press.

Line 26 : store the value for the key pressed, and any shift/CTRL/CBM logo keys.

Line 27 : if the left arrow key is pressed, then it's bye-bye program.

Line 28 : if the pound sign is pressed, turn off background rhythms.

Line 30 : if '+' key pressed, then go to routine to alter background rhythm for voice 3.

Line 31 : if F1 pressed, then go to routine to alter frequency cut-off etc. parameters.

Line 35 : if '-' key pressed, then go to routine to alter background rhythm for voice 2.

Line 60 : if RETURN key pressed, then go to routine to alter settings for all three voices.

Line 65 : if '@' pressed, then step down an octave

Line 67 : if '0' pressed, then turn off all voices.

Line 70 : if up arrow pressed, then step up an octave.

Line 75 : if '*' pressed, then toggle fake glissando mode.

Line 80 : if '=' pressed, then increase fake glissando speed.

Line 90 : get frequency relating to key pressed.

Line 95 : if no note then loop back again.

Line 100 : if numeric key pressed, then 225.

Line 105 : adjust frequency if necessary.

Line 110 : check for fake glissando.

Lines 120: if shift pressed, adjust frequency accordingly.

Lines 130-135 : calculate frequency.

Lines 140-175 : play note for relevant voices.

Line 185 : remember old frequency.

Line 190 : back for more.

Lines 195-215 : turn voices off.

Lines 225-240 : turn on/off relevant voices.

Lines 241-245 : remembering to turn on/off voices 2 and 3 via machine code program.

Lines 250-300 : on-screen display for altering voices.

Lines 315-340 : which voice.

Lines 345-375 : to which waveform.

Lines 380-410 : rest of alterations.

Lines 420-445 : turn on ring modulation.

Lines 450-500 : fake glissando effect.

Lines 505-590 : on-screen instructions.

Lines 595-715 : data and set-up voices.

Lines 800-874 : prelude to main program. Strange symbol in line 805 is the pound sign.

Lines 999-1125 : alter background rhythm for voice 2.

Lines 1999-2125 : and ditto for voice 3.

Lines 3000-3010 : Options when changing the background rhythms.

Lines 4000-4050 : driver program for background rhythms.

Lines 5000-5004 : re-set background rhythms to start at same time.

Lines 6000-6022 : options available when playing about with frequencies.

Lines 6027-6058 : altering frequency parameters.

**Notes**

As usual, the up-arrow key causes us problems in a few places, notably lines 120, 235 and 815.

The letters in italics are only so because the program has been written in lower case mode, and should be entered as shifted letters when you type the program in.

There are no graphics characters used, so there shouldn't be any great problems in deciphering the rest of the listing.

Of course, it isn't a Fairchild, but it does introduce some of the concepts involved in writing one (which should really be done totally in machine code), and that is why it is here.

# 11
# Generating Sound Effects

## Introduction

The simplest way of generating sound effects is to play about with pulse widths and filtering techniques, or to use the noise waveform and keep changing frequencies. Some of the machine code programs towards the end of this book will make life a lot easier in that respect, so by experimenting with them you'll be able to make a whole host of peculiar noises. One other popular method is to play a rapidly rising and falling note, like this:

```
10 V=54272
20 POKEV+5,9:POKEV+6,0
30 POKEV+3,0:POKEV+2,255:POKEV+4,0
40 POKEV+24,15
50 FORI=1TO100
60 POKEV+4,0:POKEV+4,65
70 POKEV+1,I
80 NEXT
90 FORI=100TO0STEP-1
100 POKEV+4,0:POKEV+4,65
110 POKEV+1,I
120 NEXT
```

which, although it looks cumbersome, produces a sound rather like a police siren.

Using the synthesiser program in the last chapter, you can also alter the ring modulation and synchronisation settings for the various voices, and this can produce some very good effects. By setting one of the background rhythms to play a simple, repetitive tune (perhaps the theme from Close Encounters, or the riff that repeats itself over and over again in Tubular Bells) and synchronising that riff with voice 1, a melody can be picked out with voice 1 and the background rhythm will move up and down the scale accordingly.

Explosions are easy enough to produce just by using the noise waveform, which is really quite a handy one in the special effects department.

To go back to purer notes for a while, the following two programs are, respectively, a program to test your scale playing, and a simple reaction timer that shows you a musical note on a keyboard which you have then to play on the keyboard as soon as possible.

## Scale practice

This doesn't use the keyboard that we've been used to in such games as Musical Simon and the like, since the scales cover a good many octaves. Far too many, in fact, to represent in the usual way. So the keyboard for this one is much the same as the one used in the synthesiser program.

The program will tell you which scale it is going to play, play it, and then you have to repeat the same scale. Six scales are built into it, although others could be added if necessary. If you get the scale perfectly right, the program moves onto the next one, but get it wrong and you have to repeat the same one all over again.

```
5 PRINTCHR$(8):DIMAN(30,2),C(30,2),G(30,2)
6 DIMD(30,2),A(30,2),E(30,2),B(30,2)
10 POKE53281,5:POKE53280,1:POKE53272,23:PRINT"[CLR
,BLK]SCALE PRACTICE.":GOSUB595
11 SC=1:GOSUB4000
12 PRINT"[2CD]I WILL PLAY THE FIRST 6 MAJOR SCALES
,ANDYOU WILL HAVE TO REPEAT ";
14 PRINT"THOSE SCALES     USING THE KEYS BELOW:"
16 PRINT"[2CD,WHT]          [RVS]Q  W  E  R  T  Y
U  I
18 PRINT"          [RVS]A  S  D  F  G  H  J  K
20 PRINT"          [RVS]Z  X  C  V  B  N  M  ,
22 PRINT"[2CD,BLK]HERE THE 'A' KEY REPRESENTS THE
KEY C:TOOBTAIN A SHARP, PRESS ";
24 PRINT"THE SHIFT KEY IN  CONJUNCTION WITH AN ALP
HABETIC KEY, AND FOR A ";
26 PRINT"FLAT PRESS THE CBM LOGO KEY IN     CONJUNC
TION WITH AN ALPHABETIC KEY.
30 PRINT"[CD]TRY USING THE KEYBOARD NOW, JUST TO G
ET THE FEEL OF IT, AND WHEN ";
32 PRINT" YOU'RE READY TOQUIT, PRESS ' '."
33 GOSUB1000:GOTO20000
34 PRINT"[CLR,BLK]SCALE PRACTICE : C MAJOR."
```

```
35 FORI=1TO29:POKES+4,0:POKES+1,C(I,1):POKES,C(I,2
):POKES+4,65:FORJ=1TO150:NEXTJ,I
36 RETURN
38 PRINT"[2CD,WHT]          [RVS]Q  W  E  R  T  Y
U  I
40 PRINT"          [RVS]A  S  D  F  G  H  J  K
42 PRINT"          [RVS]Z  X  C  V  B  N  M  .
43 PRINT"[2CD,BLK]NOW IT'S YOUR TURN: REPEAT THE S
CALE."
44 GETA$:IFA$=""THEN44
46 K=PEEK(197):PS=PEEK(653)
50 IF K=LK AND PS=LS THEN 46
52 F=N(K):LK=K:LS=PS
54 IF F=0 THEN 46
56 IF F>65535 THEN 46
58 IF PS=1 THEN F=INT(F*2^(1/12))
60 IF PS=2 THEN F=INT(F/2^(1/12))
62 F1=INT(F/256)
64 F2=F-F1*256
66 POKE S+4,0
68 POKE S,F2
70 POKE S+1,F1
72 POKE S+4,65
73 AN(A+1,1)=F1:AN(A+1,2)=F2
74 A=A+1:IFA=29THENRETURN
75 GOTO44
100 A=0:FORI=1TO29
110 IFAN(I,1)<>C(I,1)THENQ=Q+1
115 IFAN(I,2)<>C(I,2)THENQ=Q+1
120 NEXT
130 IFQ>0THENPRINT"[2CD]WRONG!":FORI=1TO2000:NEXT:
Q=0:RETURN
140 PRINT"[2CD]CORRECT!":Q=0:SC=SC+1:FORI=1TO2000:
NEXT:RETURN
150 A=0:FORI=1TO29
152 IFAN(I,1)<>G(I,1)THENQ=Q+1
155 IFAN(I,2)<>G(I,2)THENQ=Q+1
156 NEXT
157 IFQ>0THENPRINT"[2CD]WRONG!":FORI=1TO2000:NEXT:
Q=0:RETURN
158 PRINT"[2CD]CORRECT!":Q=0:SC=SC+1:FORI=1TO2000:
NEXT:RETURN
160 A=0:FORI=1TO29
162 IFAN(I,1)<>D(I,1)THENQ=Q+1
165 IFAN(I,2)<>D(I,2)THENQ=Q+1
166 NEXT
167 IFQ>0THENPRINT"[2CD]WRONG!":FORI=1TO2000:NEXT:
Q=0:RETURN
168 PRINT"[2CD]CORRECT!":Q=0:SC=SC+1:FORI=1TO2000:
NEXT:RETURN
170 A=0:FORI=1TO29
172 IFAN(I,1)<>A(I,1)THENQ=Q+1
```

```
175 IFAN(I,2)<>A(I,2)THENQ=Q+1
176 NEXT
177 IFQ>0THENPRINT"[2CD]WRONG!":FORI=1TO2000:NEXT:
Q=0:RETURN
178 PRINT"[2CD]CORRECT!":Q=0:SC=SC+1:FORI=1TO2000:
NEXT:RETURN
180 A=0:FORI=1TO29
182 IFAN(I.1)<>E(I,1)THENQ=Q+1
185 IFAN(I,2)<>E(I,2)THENQ=Q+1
186 NEXT
187 IFQ>0THENPRINT"[2CD]WRONG!":FORI=1TO2000:NEXT:
Q=0:RETURN
188 PRINT"[2CD]CORRECT!":Q=0:SC=SC+1:FORI=1TO2000:
NEXT:RETURN
190 A=0:FORI=1TO29
192 IFAN(I.1)<>B(I,1)THENQ=Q+1
195 IFAN(I,2)<>B(I,2)THENQ=Q+1
196 NEXT
197 IFQ>0THENPRINT"[2CD]WRONG!":FORI=1TO2000:NEXT:
Q=0:RETURN
198 PRINT"[2CD]CORRECT!":Q=0:SC=SC+1:FORI=1TO2000:
NEXT:RETURN
200 PRINT"[CLR,BLK]SCALE PRACTICE : G MAJOR."
202 FORI=1TO29:POKES+4,0:POKES+1,G(I,1):POKES,G(I,
2):POKES+4,65:FORJ=1TO150:NEXTJ,I
204 RETURN
210 PRINT"[CLR,BLK]SCALE PRACTICE : D MAJOR."
212 FORI=1TO29:POKES+4,0:POKES+1,D(I,1):POKES,D(I,
2):POKES+4,65:FORJ=1TO150:NEXTJ,I
214 RETURN
220 PRINT"[CLR,BLK]SCALE PRACTICE : A MAJOR."
222 FORI=1TO29:POKES+4,0:POKES+1,A(I,1):POKES,A(I,
2):POKES+4,65:FORJ=1TO150:NEXTJ,I
224 RETURN
230 PRINT"[CLR,BLK]SCALE PRACTICE : E MAJOR."
232 FORI=1TO29:POKES+4,0:POKES+1,E(I,1):POKES,E(I,
2):POKES+4,65:FORJ=1TO150:NEXTJ,I
234 RETURN
240 PRINT"[CLR,BLK]SCALE PRACTICE : B MAJOR."
242 FORI=1TO29:POKES+4,0:POKES+1,B(I,1):POKES,B(I,
2):POKES+4,65:FORJ=1TO150:NEXTJ,I
244 RETURN
450 PRINT"[CLR]THAT'S THE LOT!":PRINT"[2CD]ANOTHER
 GO (Y OR N)?"
452 FORI=1TO10:GETZ$:NEXT
454 GETZ$:IFZ$="Y"THENRUN
456 IFZ$="N"THEN20006
458 GOTO454
595 DIM N(64)
600 FOR I=0 TO 64
605 READ A:N(I)=A:NEXT
606 S=54272:POKES+3,0:POKES+2,255:POKES+5,9:POKES+
```

```
6,0:POKES+24,15:POKES+4,65
610 RETURN
620 DATA ,0,0,0,0,0,0.0
625 DATA 4,9854,4389,5.2195,4927
630 DATA 11060,0,6,11718,5530,7.2765,5859
635 DATA 13153,2463,8.14764,6577.0,3288.7382
640 DATA 16572,2930,0,17557,8286,1.4143,8779
645 DATA 0,3691,0.0,0,0,0,0.0,0,4389.0.0.0
650 DATA 0,0,0,0,0,2,0,0,3.0
655 DATA 0,8779.0,0
1000 K=PEEK(197):PS=PEEK(653)
1005 IFK=57THEN RETURN
1010 IF K=LK AND PS=LS THEN 1000
1020 F=N(K):LK=K:LS=PS
1030 IF F=0 THEN 1000
1040 IF F>65535 THEN 1000
1050 IF PS=1 THEN F=INT(F*2^(1/12))
1060 IF PS=2 THEN F=INT(F/2^(1/12))
1070 F1=INT(F/256)
1080 F2=F-F1*256
1090 POKE S+4,0
1110 POKE S,F2
1120 POKE S+1,F1
1122 POKE S+4,65
1125 GOTO1000
4000 DATA8,147,9,159.10.205.11.114.12.216.14.107.1
6,47,17,37,19,63,21.154
4002 DATA22,227.25,177.28,214.32,94.34.75
4004 DATA32,94,28,214,25,177,22,227,21,154.19,63.1
7,37,16,47,14,107.12,216
4006 DATA11,114,10,205.9,159,8,147
4008 FORI=1TO29:READX,Y:C(I,1)=X:C(I,2)=Y:NEXT
4010 DATA12,216,14,107,16,47,17,37,19.63,21,154.22
,227,25,177,28.214,32,94
4012 DATA34,75,38,126,43,52,48,126,51.97
4014 DATA48,126,43,52.38,126,34,75.32.94,28,214,25
,177,22,227,21,154,19,63,17,37
4016 DATA16,47,14,107,12,216
4018 FORI=1TO29:READX,Y:G(I,1)=X:G(I,2)=Y:NEXT
4020 DATA9,159,10,205,11,114,12,216,14,107,16.47,1
8,42,19,63,21,154,24,63
4022 DATA25,177,28,214,32,94,34,75,38,126
4024 DATA34,75,32,94,28,214,25,177,24,63,21,154,19
,63,18,42,16,47,14,107,12,216
4026 DATA11,114,10,205,9,159
4028 FORI=1TO29:READX,Y:D(I,1)=X:D(I,2)=Y:NEXT
4030 DATA14,107,16,47,17,37,19,63,21,154,22,227,25
,177,28,214,32,94,36,85
4032 DATA38,126,43,52,48,126,54,111,57,172,54,111,
48,126,43,52,38,126,36,85
4034 DATA32,94,28,214,25,177,22,227,21,154,19,63,1
7,37,16,47,14,107
```

```
4038 FORI=1TO29:READX,Y:A(I,1)=X:A(I,2)=Y:NEXT
4040 DATA10,205,11,114,12,216,14,107,16,47,17,37,1
9,63,21,154,22,227,25,177
4042 DATA28,214,32,94,36,85,40,199,43,52,40,199,36
,85,32,94,28,214,25,177
4044 DATA22,227,21,154,19,63,17,37,16,47,14,107,12
,216,11,114,10,205
4048 FORI=1TO29:READX,Y:E(I,1)=X:E(I,2)=Y:NEXT
4050 DATA16,47,17,37,19,63,21,154,22,227,25,177,28
,214,32,94,36,85,40,199,43,52
4052 DATA48,126,54,111,57,172,64,188,57,172,54,111
,48,126,43,52,40,199,36,85
4054 DATA32,94,28,214,25,177,22,227,21,154,19,63,1
7,37,16,47
4058 FORI=1TO29:READX,Y:B(I,1)=X:B(I,2)=Y:NEXT
4159 RETURN
20000 ONSCGOSUB34,200,210,220,230,240,450
20002 GOSUB38
20004 ONSCGOSUB100,150,160,170,180,190
20005 GOTO20000
20006 PRINT"[CD]YOU'VE PLAYED THEM ALL ...."
20008 END
```

**Explanation**

A lot of this will be familiar from other programs, so only the briefest of notes will be given here.

Lines 5-6 : Set up arrays to hold note information for each scale.

Lines 12-33 : On screen instructions.

Lines 34-36 : Play scale of C major.

Lines 38-75 : Usual routine for playing a note from the keyboard.

Lines 100-140 : Check every note you played against every note in the scale. If you've got it right, go on to the next one, but if not go back to the same scale again.

Lines 150-158 : Ditto for next scale.

Lines 160-198 : And again, for all other scales.

Lines 200-244 : Playing each of the scales in turn.

Lines 450-458 : Another game?

Lines 598-655 : Reading data for keys.

Lines 1000-1125 : Usual note playing routine.

Lines 4000-4159 : Data, and reading of it, for each note in each scale.

Lines 20000-20008 : Control of program flow is all done from this little routine.

## Reaction timer

With this program you're given the usual musical keyboard as used in Musical Simon, and after a random interval of time a note will appear on the screen, and be played as well for audio recognition. Then, as fast as you can, you must repeat the note. You're marked after every ten notes, told how many you got right and how many you got wrong, and the average amount of time it took you to press the relevant keys.

```
5 GOSUB30000:GOSUB10000
9 GOTO200
10 PRINT"[SSP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF]";
11 PRINT"[SP,RVS,2SP,OFF,4SP,RVS,2SP,OFF]"
12 REM LINES 10 AND 11 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
20 PRINT"[SSP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF]";
21 PRINT"[SP,RVS,2SP,OFF,4SP,RVS,2SP,OFF]"
22 REM LINES 10 AND 11 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
30 PRINT"[SSP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF]";
31 PRINT"[SP,RVS,2SP,OFF,4SP,RVS,2SP,OFF]"
32 REM LINES 10 AND 11 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
40 PRINT"[SSP,RVS,BLK,2SP,OFF,SP,RVS,2SP,OFF,4SP,R
VS,2SP,OFF,SP,RVS,2SP,OFF]";
41 PRINT"[SP,RVS,2SP,OFF,4SP,RVS,2SP,OFF]"
42 REM LINES 10 AND 11 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
50 PRINT"[6SP,RVS,WHT,2SP,BLK,2SP,WHT,SP,BLK,2SP,W
HT,SP,CBMM,2SP,BLK,2SP,WHT,SP,]";
51 PRINT"[BLK,2SP,WHT,SP,BLK,2SP,WHT,SP,CBMM,2SP,B
LK,2SP]"
```

```
52 REM LINES 50 AND 51 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
60 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM,]";
61 PRINT"[2SP,CBMM,2SP,CBMM]"
62 REM LINES 60 AND 61 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
70 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM,]";
71 PRINT"[2SP,CBMM,2SP,CBMM]"
72 REM LINES 70 AND 71 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
80 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM,]";
81 PRINT"[2SP,CBMM,2SP,CBMM]"
82 REM LINES 80 AND 81 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
90 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,2
SP,CBMM,2SP,CBMM,2SP,CBMM,]";
91 PRINT"[2SP,CBMM,2SP,CBMM]"
92 REM LINES 90 AND 91 JOIN TOGETHER TO FORM ONE L
INE ON THE 64
100 PRINT"[6SP,RVS,WHT,2SP,CBMM,2SP,CBMM,2SP,CBMM,
2SP,CBMM,2SP,CBMM,2SP,CBMM,]";
101 PRINT"[2SP,CBMM,2SP,CBMM]"
102 REM LINES 100 AND 101 JOIN TOGETHER TO FORM ON
E LINE ON THE 64
110 RETURN
200 POKES+21,0:PRINT"[CLR]NOTE COMING UP:[4CD]":GO
SUB10:GG=GG+1
201 PRINT"[HOME,3CD,WHT]          #  #     #  #  #
    #"
202 PRINT"[HOME,4CD,WHT]         C  D     F  G  A
   C"
203 PRINT"[HOME,15CD]       C  D  E  F  G  A  B  C
"
204 FORI=1TOINT(RND(.5)*1000+1000):NEXT
205 KE$(1)=MID$(KE$,INT(RND(.5)*13+1),1)
206 FORI=1TO10:GETN$:NEXT
207 POKEV+4,0
208 FORI=1TO14:IFKE$(1)=MID$(KE$,I,1)THENGOSUB5000
0:I=14
209 NEXTI:POKEV+4,65:POKES+21,1:T=TI
210 PRINT"[2CD]GO"
220 P=0:GETN$:IFN$=""THEN220
225 FORI=1TO14:IFN$=MID$(KE$,I,1)THENP=1
226 NEXTI:IFP=0THEN220
230 AN$=N$
232 FORI=1TOLEN(KE$)
234 IFN$=MID$(KE$,I,1)THENPOKEV+4,0:POKEV+4,65:GOS
UB50000
235 NEXT
```

```
240 T1=TI:PRINT"[CD]TIME TAKEN = ";(T1-T)/60;" SEC
ONDS."
260 IFAN$<>KE$(1)THEN300
280 PRINT"[CD]CORRECT!";FORJ=0TO1000:NEXT:GR=GR+1:
TR=TR+(T1-T)/60:IFGG=10THEN321
286 GOTO200
300 PRINT"[CD,RVS]WRONG!   ";:FORJ=0TO1000:NEXT:GW=
GW+1:TW=TW+(T1-T)/60:IFGG=10THEN321
320 GOTO200
321 PRINT"[CLR]RESULTS TIME:"
322 PRINT"[CD]GUESSES RIGHT = ";GR
323 PRINT"GUESSES WRONG = ";GW
324 PRINT"AVERAGE TIME  = ";TR/GR;" SECONDS"
325 PRINT"FOR RIGHT GUESSES.
326 PRINT"AVERAGE TIME = ";TW/GW;" SECONDS
327 PRINT"FOR WRONG GUESSES."
328 PRINT"[2CD]ANOTHER GO (Y OR N)?";:POKES+1,0:POK
ES,0
329 TW=0:TR=0:GR=0:GW=0:GG=0
330 GETG$:IFG$="Y"THEN200
340 IFG$="N"THENPRINT"[2CD]BYE.":END
350 GOTO330
9999 END
10000 POKE53281,11:POKE53280,11:POKE53272,23:PRINT
"[CLR,BLK,RVS]WELCOME TO REACTION"
10002 PRINT"[CD]THIS GAME WILL TEST YOUR REACTIONS
, AND YOUR ABILITY TO RECOGNISE";
10004 PRINT" AND PLAY A     MUSICAL NOTE."
10006 PRINT"[CD]THE COMPUTER, AFTER A RANDOM INTER
VAL OFTIME, WILL GENERATE A ";
10008 PRINT"MUSICAL (?!) NOTE AND YOU HAVE TO IDEN
TIFY AND PLAY THAT  NOTE ";
10010 PRINT"AS QUICKLY AS POSSIBLE.":GOSUB10:GOSUB
20000
10012 PRINT"[CLR,RVS]REACTION"
10014 PRINT"[CD]THE GAME WILL BE PLAYED USING THE
KEYS  INDICATED BELOW."
10016 PRINT"[HOME,4CD]":GOSUB10
10026 PRINT"[HOME,5CD,WHT]          W  E     T  Y  U
     O"
10028 PRINT"[HOME,15CD]        A  S  D  F  G  H  J
 K"
10030 PRINT"[2CD]TRY IT NOW, JUST TO GET THE FEEL
OF THE KEYBOARD.  WHEN YOU'RE FED";
10032 PRINT" UP, PRESS     'SPACE' TO GET THE SHOW
ROLLING."
10040 GETA$:IFA$=""THEN10040
10045 IFA$=" "THEN10900
10050 FORI=1TOLEN(KE$)
10060 IFA$=MID$(KE$,I,1)THENPOKEV+4,0:POKEV+4,65:G
OSUB50000
10070 NEXT
```

```
10090 GOTO10040
10900 PRINT"[CD]YOU'LL BE MARKED AFTER EVERY TEN G
OES.":FORJ=1TO1500:NEXT
10999 POKES+1,0:POKES,0:RETURN
20000 PRINT"[2CD,RVS]PRESS 'SPACE' TO CONTINUE."
20002 FORI=1TO10:GETSP$:NEXT
20004 GETSP$:IFSP$<>" "THEN20004
20006 RETURN
30000 V=54272:POKEV+3,255:POKEV+5,9:POKEV+6,0:POKE
V+4,65:POKEV+24,15
30001 DIMNO(28),KE$(255),AN$(255),SP(28)
30002 DATA4,73,4,208,5,103,5,185,6,108,7,53,8,23,8
,147,4,139,5,25,6,16,6,206
30004 DATA7,163,9,21
30006 FORI=1TO28:READNO(I):NEXT
30008 KE$="ASDFGHJKWETYUO"
30010 S=53248
30012 FORI=0TO62:READA:POKE832+I,A:NEXT
30014 POKE2040,13:POKES+23,0:POKES+29,0:POKES+28,0
:POKES+39,5:X=0:Y=0
30015 POKES+1,Y:POKES,X:POKES+21,1
30016 FORI=1TO28:READSP(I):NEXT
30018 RETURN
40001 DATA0,0,0,0,0,0,0
40002 DATA32,0,0,48,0,0,56
40003 DATA0,0,60,0,0,38,0
40004 DATA0,34,0,0,34,0,0
40005 DATA36,0,0,32,0,0,32
40006 DATA0,0,32,0,0,32,0
40007 DATA3,224,0,7,224,0,15
40008 DATA240,0,15,240,0,7,224
40009 DATA0,1,128,0,0,0,0
41000 DATA 73,135,97,135,121,135,145,135,169,135,1
93,135,217,135,241,135
41002 DATA 86,95,110,95,158,95,182,95,206,95,255,9
5
50000 POKEV+1,NO(I+I-1):POKEV,NO(I+I):POKES+1,SP(I
+I):POKES,SP(I+I-1):RETURN
```

**Explanation**

Most important of all! Lines 322 and 323 (just spotted this!) need altering slightly, to read as follows:

```
322 PRINT"[CD]GUESSES RIGHT = ";GR:IFGR=0THENGR=1

323 PRINT"GUESSES WRONG = ";GW:IFGW=0THENGW=1
```

Otherwise, a brilliantly successful session or abysmally bad session

(guess which one I had) will produce a division by zero error in line 324 or line 326. Sorry about that.

Lines 10-110 : Usual keyboard.

Lines 200-209 : Display keyboard and, after a random interval of time, play and display a note.

Lines 210-240 : Get player's note, and take time.

Lines 260-320 : Right or wrong?

Lines 321-350 : Player marked, and another game is requested.

Lines 10000-20006 : Display instructions and keyboard, and let player play on keyboard for a while.

Lines 30000-30018 : Read in sprite and note data.

Lines 40000-41002 : Data for sprite and note.

Line 50000 : Little routine to play and display note.

Once you've finished with that one, it's time for some games with sound in Chapter 12.

# 12
# Musical Games Programming

## Introduction

Using sound with games is very much a matter of personal taste. Some people like background rhythms playing all the time, others don't, so if you do use music always include an option to turn it off. This shouldn't take away the 'sound effects' part of the game, merely the tune. How many of us have, after just a couple of seconds, turned the sound off when playing Manic Miner (not that this particular game is the only offender).

Accordingly, the two games listed here do include a tune (loaded in the driver programs headed Pestart and Mustart respectively), but also give you the option to turn that tune off.

## Pestroy

In other words, destroy the pests that come after you. A number of interesting routines lie in this game, including scrolling the screen character by character left and right, playing that background tune, setting up a whole host of user-defined characters, and so on. Play using the joystick in port two, and if you want the starter program to load everything from tape rather than disk, alter lines 0 and 1170 in that same driver program.

PESTART

```
0 IFPEEK(49219)<>65THENLOAD "BOOGIETIME",8,1
1160 V=54272:POKEV+17,255:POKEV+19,142:POKEV+20,15
0
1161 POKEV+24,15:POKE49216,10:POKE49218,0:POKE4921
9,65:POKE251,178:POKE252,192
1162 POKE253,178:POKE254,192
1163 POKEV+10,255:POKEV+12,142:POKEV+13,150
1164 POKE51264,10:POKE51266,0:POKE51267,65:POKE247
,178:POKE248,200
```

```
1166 POKE249,178:POKE250,200:SYS49152
1170 PRINT"[CLR,2CD]LOAD";CHR$(34);"PESTROY";CHR$(
34);",8[4CD]":PRINT"RUN[HOME]";
1175 POKE198,2:POKE631,13:POKE632,13:END

READY.



PESTROY


15 PRINTCHR$(8)
20 POKE56,48:POKE52,48:CLR:E=5
25 POKE53281,0:POKE53280,0:PRINT"[CLR,YEL]"TAB(17)
"PESTROY":S=0:HS=0:CO=55296:SC=1024
26 PRINT"[2CD]SHOOT EVERYTHING AND ANYTHING ...
30 PRINT"[2CD]MOVE USING JOYSTICK IN PORT 2:"
32 PRINT"TO TURN BACKGROUND TUNE OFF, PRESS F1
33 PRINT"[CD]TO TURN IT ON, PRESS F3
34 PRINT"[2CD]NOW PLEASE HANG ON WHILE I READ A FE
W   NEW CHARACTERS IN."
38 GOSUB1100
40 GOSUB30000
42 GOSUB40000
43 GOSUB900
45 GOTO100
50 PRINT"[HOME]                    "
60 PRINT"[HOME,WHT]SCORE ";S;TAB(20)"HIGH SCORE ";
HS:RETURN
100 GOSUB1000:GOSUB700
102 GOSUB50
103 SP=400
110 A=INT(RND(.5)*3+33):B=INT(RND(.5)*2+38)
115 C=INT(SP/40):C=C*40+39:IFC>680THENC=679
116 POKEC+SC,A:IFINT(RND(.5)*10)>5THENPOKEC+SC,B
120 POKESC+SP,36:POKESC+SP+1,37:POKECO+SP,7:POKECO
+SP+1,7
121 IFG=1ANDTI-TO>120THENGOSUB50:G=0
125 POKESC+SP,32:POKESC+SP+1,32:SYS828
126 IFPEEK(SC+SP+2)<>32THEN800:REM END OF GAME
127 Z=PEEK(197):IFZ=4THENPOKE49219,64:POKE51267,64
128 IFZ=5THENPOKE49219,65:POKE51267,65
130 Z=PEEK(56320):S2=56320
132 IFZ=123ORZ=107THENSP=SP-1:POKES1+1,3:IFSP/40=I
NT(SP/40)THENSP=SP+1:GOTO120
134 IFZ=119ORZ=103THENSP=SP+1:POKES1+1,5:IF(SP-23)
/40=INT((SP-23)/40)THENSP=SP-1:GOTO120
136 IFZ=126ORZ=110THENSP=SP-40:IFSP<40THENSP=SP+40
:GOTO120
138 IFZ=125ORZ=109THENSP=SP+40:IFSP>960THENSP=SP-4
0:GOTO120
```

```
140 IFZ=111THENPOKESC+SP,36:POKESC+SP+1,37:POKECO+
SP,7:POKECO+SP+1,7:GOTO600:GOTO120
145 IFINT(RND(.5)*10)>5THEN110
150 GOTO120
600 FORI=1TO10:IFPEEK(SC+SP+I+1)=78THENFF=1
601 NEXT:IFFF=1THENFF=0:GOTO120
602 E=E+1:IFE/5=INT(E/5)THENPRINT"[HOME]OVERHEATED
":TO=TI:G=1:GOTO650
603 IFG=1ANDTI-TO<120THEN120
604 GOSUB50:FORI=1TO10:IFPEEK(SC+I+1+SP)<>32THENS=
S+10:POKES1+4,129:FORK=1TO50:NEXT:POKES1+4,65
605 POKESC+SP+I+1,45:POKECO+SP+I+1,9
606 POKES1+1,30:NEXT
607 FORI=1TO10:POKESC+SP+I+1,32:NEXT
608 GOSUB50:POKES1+1,4:GOTO120
650 FORI=1TO5:FORJ=0TO20:POKES1+1,J:NEXTJ,I:POKES1
+1,4:GOTO120
700 S1=54272:POKES1+3,12:POKES1+5,40:POKES1+6,146:
POKES1+24,15:POKES1+4,65
702 POKES1+1,4:RETURN
800 PRINT"[CLR]YOU BLEW IT WITH A SCORE OF ":PRINT
"[CD,CR]";S
805 POKES1+4,129:FORI=250TOOSTEP-1:POKES1+1,I:NEXT
:POKES1+24,0
810 IFS>HSTHENPRINT"[2CD]BUT AT LEAST YOU DID GET
THE NEW HIGH   SCORE.":HS=S:S=0
820 PRINT"[2CD]OTHER GAME (Y OR N)?"
825 GETAG$:IFAG$="Y"THENPOKES1+24,15:PRINT"[CLR]":
GOTO43
826 IFAG$="N"THENPRINT"[CLR,2CD]BYE.":END
827 GOTO825
899 END
900 REM DRAW MOUNTAIN RANGE
914 PRINT"[CLR,23CD]";:
918 RUN LINES 919,920 AND 921 TOGETHER TO FORM ONE
 CONTINUOUS LINE
919 REMS\ REPRESENTS SHIFTED POUND, C* REPRESENTS
CBM KEY AND *.
920 PRINT"[GRN,RVS,S\,CU,S\,CU,S\,CU,S\,C*,CD,C*,C
D,C*,S\,CU,S\,CU,S\,C*,CD,C*]";
921 PRINT"[CD,C*,S\,CU,S\,CU,S\,CU,S\,C*,CD,C*,CD,
C*,CD,C*,CD,C*,S\,CU,S\,CU,S\,C*]";
922 PRINT"[CD,C*,S\,CU,S\,CU,S\,C*,CD,C*,CD,C*,S\,
CU,S\,CU,S\,C*,CD,C*,CD,C*,CD,C*]"
923 PRINT"[4CU,16CR,RVS,2SP]"
924 PRINT"[RVS,3CR,2SP,4CR,2SP,4CR,4SP,10CR,2SP]";
925 PRINT"[RVS,2CR,4SP,2CR,4SP,2CR,7SP,4CR,2SP,2CR
,4SP,4CR,2SP]"
926 PRINT"[RVS,CR,20SP,2CR,10SP,2CR,4SP]";
930 FORI=1984TO2023:POKEI,160:NEXT
932 FORI=56056TO56295:POKEI,5:NEXT
940 PRINT"[HOME]
```

```
945 RETURN
1000 REM SET UP COLOUR FOR CHARACTERS ON SCREEN
1002 FORI=40TO139:POKECO+I,1:NEXT
1004 FORI=160TO279:POKECO+I,4:NEXT
1006 FORI=280TO399:POKECO+I,5:NEXT
1008 FORI=400TO519:POKECO+I,6:NEXT
1010 FORI=520TO679:POKECO+I,10:NEXT
1012 RETURN
1089 END
1100 POKE56333,127
1105 POKE1,51
1110 FORX=0TO1024
1120 POKE12288+X,PEEK(53248+X)
1130 NEXT
1140 POKE1,55
1150 POKE56333,129
1160 POKE53272,29
1170 RETURN
30000 DATA24,28,30,27,25,120,240,96
30001 DATA48,60,54,51,60,54,227,224
30002 DATA6,103,110,119,110,246,102,224
30003 DATA224,240,31,15,15,31,240,224
30004 DATA0,0,224,255,255,224,0,0
30006 DATA63,33,33,33,33,231,231,198
30007 DATA192,192,248,252,204,216,240,224
30008 FORI=0TO55:READA:POKE12552+I,A:NEXT:PRINT"[C
LR]":RETURN
40000 DATA169,40,162,24,133,87,169,4,133,88,160,0,
177,87,133,89,200,177
40002 DATA87,136,145,87,200,152,201,39,208,244,165
,89,145,87,165,87,24,105
40004 DATA40,133,87,144,2,230,88,202,208,220,96,-1
40006 A=828
40008 READB:IFB<>-1THENPOKEA,B:A=A+1:GOTO40008
40010 RETURN

READY.
```

## Musicman

Variations on a theme here, as Musicman features a number of musical aliens chasing around the usual maze gobbling dots and eating everything they come across.

Again, if you want to load from tape rather than disk, alter lines 0 and 1170. The music for both these games follows the main listings.

MUSTART

```
0 IFPEEK(49219)<>65THENLOAD "YELLOW".8,1
1160 V=54272:POKEV+17,255:POKEV+19,142:POKEV+20,15
0
1161 POKEV+24,15:POKE49216,10:POKE49218,0:POKE4921
9,65:POKE251,178:POKE252,192
1162 POKE253,178:POKE254,192
1163 POKEV+10,255:POKEV+12,142:POKEV+13,150
1164 POKE51264,10:POKE51266,0:POKE51267,65:POKE247
,178:POKE248,200
1166 POKE249,178:POKE250,200:SYS49152
1170 PRINT"[CLR,2CD]LOAD";CHR$(34);"MUSICMAN";CHR$
(34);",8[4CD]":PRINT"RUN[HOME]";
1175 POKE198,2:POKE631,13:POKE632,13:END

READY.
```

MUSICMAN

```
90 POKE53280,7:POKE53281,0:PRINT"[YEL]
100 DIM C%(255),C(8),O(8),RX(4),RY(4)
110 O=50176:S1=54272:POKES1+2,255:POKES1+5,9:POKES
1+6,0:POKES1+4,65:POKES1+24,15
120 C%(60)=-1:C%(62)=-1:C%(22)=-1:C%(1)=-1:C%(93)=
-1:C%(64)=-1
130 FORX=2TO8STEP2:READC(X):NEXTX:DATA131,129,128,
130
140 FORX=2TO8STEP2:READO(X):NEXTX:DATA93,64,64,93
160 C(0)=C(6):O(0)=O(6)
200 GOSUB20000
210 T(2)=L:T(4)=-1:T(6)=1:T(8)=-L
220 N8=O+11*L:N9=N8+39
300 S=0:LV=.90:S6=10:CC=2
399 REM THE + SYMBOL REPRESENTS PRESSING CBM AND +
 AT THE SAME TIME.
400 A=0:PRINT"[CLR,BLU]++++++++++++++++++++++++++
++++++++++
410 PRINT"+.................++.................
420 PRINT"+.++++.++++.++++++.++.++++++.++++.++++.
430 PRINT"+.+....+....+......++......+....+....+.
440 PRINT"+.+.++.+.++.+.++++.++.++++.+.++.+.++.+.
450 PRINT"+.+.++.+.++.+....+.++.+....+.++.+.++.+.
460 PRINT"+............++...++...++............
470 PRINT"+.++++++.++++.++.+.++.+.++.++++.++++++.
480 PRINT"+.+......+....*+........+*....+......+.
490 PRINT"+...++++.+.+++.+.++++++.+.+++.+.++++...
```

```
500 PRINT"+++++....+.....................+....++++
510 PRINT"     .++++++.+++++++++++++++.++++++.
520 PRINT"+++++....+.....................+....++++
530 PRINT"+...++++.+.+++.+.++++++.+.+++.+.++++...
540 PRINT"+.+......+....*+........+*....+......+.
550 PRINT"+.++++++.++++.++.+.++.+.++.++++.++++++.
560 PRINT"+............++...++...++............
570 PRINT"+.+.++.+.++.+....+.++.+....+.++.+.++.+.
580 PRINT"+.+.++.+.++.+.++++.++.++++.+.++.+.++.+.
590 PRINT"+.+....+....+......++......+....+....+.
600 PRINT"+.++++.++++.++++++.++.++++++.++++.++++.
610 PRINT"+..................++.................
620 PRINT"++++++++++++++++++++++++++++++++++++++++
625 POKEO+8*40+14,133:POKES1+1024+8*40+14,4
626 POKEO+8*40+25,133:POKES1+1024+8*40+25,4
627 POKEO+14*40+25,133:POKES1+1024+14*40+25,4
628 POKEO+14*40+14,133:POKES1+1024+14*40+14,4
630 REM
640 NX=39:FORNY=0TO22:IFNY<>11THENPOKEO+NX+L*NY,10
2:POKENX+L*NY+S1+1024,6
650 NEXTNY
655 IFLL>0THEN730
660 ZX=132:FORY=1TO4
670 NX=INT(RND(1)*37+2)
680 NY=INT(RND(1)*21+2)
690 Z=O+NX+L*NY:IFPEEK(Z)<>46GOTO670
700 B(Y)=PEEK(Z):POKEZ,ZX:POKEZ+5120,CC
710 RX(Y)=NX:RY(Y)=NY
720 NEXTY
730 N=O+L+1:NO=N:POKEN,C(6):POKEN+5120,7
1000 P=PEEK(56320):IFP=127THEN1040
1003 R1=P
1005 IFR1=111THEN3000
1011 IFR1=123THENR1=4:GOTO1020
1012 IFR1=119THENR1=6:GOTO1020
1014 IFR1=126THENR1=8:GOTO1020
1015 IFR1=125THENR1=2:GOTO1020
1016 GOTO1000
1020 IFT(R1)=0GOTO1000
1030 R=R1:R2=T(R)
1040 NO=N+R2
1050 N1=PEEK(NO)
1060 IFNO=N8THENNO=NO+38:GOTO1050
1070 IFNO=N9THENNO=NO-38:GOTO1050
1080 IFN1=102THENNO=N:N1=32
1090 IFN1=133THENGOSUB5000
1100 IFN1=46THENS=S+5:A=A+1:GOSUB25000
1110 IFN1=132GOTO4000
1120 IFN1=134THENGOSUB5800:GOTO1040
1130 S8=66:IFN1=46THENS8=125
1160 S9=S8:Q=N-O:PY=INT(Q/L):PX=Q-PY*L
1170 FORY=1TO4
```

```
1180 IFRND(1)<LVGOTO1260
1190 RX=RX(Y):IFRX=0GOTO1260
1200 NX=RX+SGN(PX-RX):NY=RY(Y)+SGN(PY-RY(Y))
1210 Z=0+NX+L*NY:C%=PEEK(Z):IFC%=ZXGOTO1260
1220 POKE0+RX(Y)+L*RY(Y),B(Y):B(Y)=32:POKE0+RX(Y)+
RY(Y)*L+5120.6
1225 IFC%(C%)GOTO3950
1230 IFZ=NOANDZX=216THENGOSUB6000:GOTO1260
1240 POKEZ,ZX:POKEZ+5120,CC:S9=253:B(Y)=C%
1250 RX(Y)=NX:RY(Y)=NY
1260 NEXTY
1270 IFZX=134THENGOSUB7000
1280 POKEN,32:POKEN0,C(R):POKEN0+5120,7
1290 POKES1+4,0:POKES1+4,65:POKES1+1,6:POKES1,80
1300 POKES1+1,INT(S8/10):FORX=1TOS6:NEXTX:POKES1+4
,0:POKES1+4,65
1305 POKES1+1,INT(S9/10):FORX=1TOS6:NEXT
1310 POKES1+1,0
1320 POKEN0,0(R):POKEN0+5120,7
1330 N=N0
1340 IFA<>448GOTO1000
2000 LV=LV*LV:S6=S6/2+2
2010 F=F+1
2020 POKES1+4,0:POKES1+4,65:POKES1+1,6:POKES1,80:F
ORX=0TO20:POKES1,X:NEXTX
2030 FORX=100TO70STEP-1:POKES1+1,X:NEXTX
2040 GOTO400
3000 IFBT=0THENPOKE49219,64:POKE51267,64:BT=1:GOTO
1000
3002 POKE49219,65:POKE51267,65:BT=0:GOTO1000
3950 IFZX=134THENGOSUB6000:GOTO1040
3999 REM SUICIDE
4000 FORI=0TO9:Y(I)=PEEK(50416+I):Z(I)=PEEK(50376+
I):NEXT
4001 FORK=1TO10:POKES1+4,0:POKES1+4,129
4002 POKES1+1,6:POKES1,80:FORX=80TO130STEP2:POKES1
+1,X:NEXTX:POKES1,0
4015 IFK/2<>INT(K/2)THEN4030
4020 PRINT"[HOME,5CD,RED]GOT YOU!":POKEN,214:POKES
1+1,58:GOTO4040
4025 POKES1+4,0:POKES1+4,65:POKES1+1,6:POKES1,80:P
OKES1+1,58:GOTO4040
4030 PRINT"[HOME,5CD,GRN]GOT YOU!":POKEN,86:POKES1
+1,48
4035 NEXTK
4040 NEXTK:PRINT"[BLU]
4041 POKES1+1,0:POKES1+4,65
4042 LL=LL+1:IFLL=3THENPP=PP+1:GOTO4045
4043 GOSUB25000:GOTO400
4045 PRINT"[CLR]"TAB(11);"[YEL]*** MUSICMAN ***":L
L=0:POKES1+24,0
4050 PRINT"[3CD,3CR]YOUR SCORE WAS ";S;"[CD]"
```

```
4051 IFHS=0THENHS=S:GOTO4054
4052 IFS>HSTHENPRINT"[CD]    A NEW HIGH SCORE! [CD]
":HS=S:GOTO4054
4053 PRINT"[CD]    HIGH SCORE:":HS
4054 PRINT"[CD,3CR]GUARDS CAUGHT:":CG:""
4060 PRINT"[3CD]DO YOU WANT TO PLAY AGAIN?"
4070 GETW$:IFW$=""THEN4070
4080 IFW$="N"THENPRINT"[CLR,2CR]BYE[HOME]"::POKE63
1,13:POKE198,1:END
4090 IFW$<>"Y"THEN4070
4100 POKES1+24,15:GOTO300
4999 REM EAT A PILL
5000 ZX=134:FORY=1TO4
5010 NX=RX(Y):NY=RY(Y):IFNX=0THEN530
5020 POKEO+NX+L*NY,ZX:POKEO+NX+L*Y+5120,2
5030 NEXTY
5032 POKES1+4,0:POKES1+4,65:FORI=0TO100:POKES1+1,I
:NEXT:FORI=100TO0STEP-1:POKES1+1,I:NEXT
5035 TO=TI+600:CC=5
5040 RETURN
5799 REM CHOMP A GHOST
5800 FORY=1TO4:IFNO=RX(Y)+RY(Y)*L+0GOTO5830
5820 NEXTY
5830 POKENO,B(Y):POKENO+5120,6
5999 REM: WIPE A GHOST
6000 S=S+100
6005 GOSUB25000
6010 CG=CG+1:POKES1+4,0:POKES1+4,65
6012 POKES1+1,6:POKES1+4,129:POKES1,80:FORX=10TO10
0STEP2:POKES1+1,X:NEXTX:POKES1,0:POKES1+4,33
6016 RX(Y)=0
6060 RETURN
7000 IFTI>TOGOTO7100
7010 IFTO-TI>60GOTO7160
7020 FORY=1TO4
7030 RX=RX(Y):IFRX<>0GOTO7070
7040 NX=Y+18:NY=11:Z=O+NX+L*NY:IFPEEK(Z)=196GOTO070
60
7050 POKEZ,196:POKEZ+5120,2:GOTO7070
7060 POKEZ,102:POKEZ+5120,6
7070 NEXTY
7080 POKES1+1,6:POKES1,80:FORX=50TO80:POKES1,90-X:
NEXTX
7090 POKES1+1,0:RETURN
7100 ZX=132:FORY=1TO4
7110 NX=RX(Y):NY=RY(Y):IFNX<>0GOTO7140
7120 NX=Y+18:RX(Y)=NX:NY=11:RY(Y)=NY:B(Y)=102
7140 POKEO+NX+L*NY,ZX:POKEO+NX+L*NY+5120,6
7150 NEXTY:CC=2
7160 RETURN
7975 POKES1+4,0:POKES1+4,65
20000 PRINT"[CLR]"TAB(11);"[YEL]*** MUSICMAN ***"
```

```
20001 L=40:IFCP=1THEN20010
20002 PRINT"[2CD]PLEASE WAIT A LITTLE WHILE.":GOSU
B29000
20010 PRINTTAB(1):"[CD]RIGINALLY CREATED BY PAUL G
UMMERSALL"
20015 PRINTTAB(4):"[2CD]CODING CHANGES BY JIM BUTT
ERFIELD"
20018 PRINT"[CD.3CR]ADAPTED FOR THE CBM64 BY YOURS
 TRULY"
20020 PRINTTAB(6)"[3CD.SHIFTU.26SHIFT*.SHIFTI]"
20030 PRINTTAB(6)"[SHIFT-] DO YOU WANT INSTRUCTION
S [SHIFT-]"
20035 PRINTTAB(6)"[SHIFT-]          YES OR NO
  [SHIFT-]"
20040 PRINTTAB(6)"[SHIFTJ.26SHIFT*.SHIFTK]"
20050 GETW$:IFW$=""THEN20050
20060 IFW$="N"THENRETURN
20070 IFW$="Y"THEN20100
20080 GOTO20050
20100 PRINT"[CLR]"TAB(11):"*** MUSICMAN ***"
20110 PRINT"[3CD]     WELCOME TO THE GAME OF MUSIC
MAN."
20120 PRINT"[CD]   IN THIS GAME YOU TRY TO EAT AS
MANY"
20130 PRINT"   OF THE LITTLE DOTS IN THE MAZE AS "
20140 PRINT"   YOU CAN. THIS MAY SOUND EASY BUT YO
U"
20150 PRINT"   WILL BE CHASED BY GUARDS THAT LOOK"
20160 PRINT"   LIKE THIS: [RVS.RED]D[OFF.YEL]. SO
KEEP AWAY FROM THEM."
20170 PRINT"   USE JOYSTICK IN PORT 2 TO MOVE: "
20230 PRINT"[CD]PRESS ANY KEY TO CONTINUE"
20240 GETW$:IFW$=""THEN20240
20250 PRINT"[CLR]"TAB(11)"*** MUSICMAN ***"
20260 PRINT"[2CD]   YOU HAVE A CHANCE OF CAPTURING
 THE"
20270 PRINT"   GUARDS. YOU HAVE TO EAT ONE OF THE"
20280 PRINT"   ENERGIZERS THAT LOOK LIKE THIS:[RVS
,PUR]E[YEL]"
20285 PRINT"   WHEN YOU DO, THE GUARDS APPEAR
20290 PRINT"   LIKE THIS:[RVS,GRN]F[OFF.YEL].  NOW
 YOU CAN GET THEM
20295 PRINT"   AND GAIN 100 POINTS EACH.
20296 PRINT
20297 PRINT"   BUT WATCH IT:  THE ENERGIZERS LAST
20300 PRINT"   ONLY TEN SECONDS: THEN THE
20301 PRINT"   GUARDS ARE BACK AND DANGEROUS.
20302 PRINT"[CD]   OH YES, THEY CAN ALSO WALK THRO
UGH      THE WALLS!"
20304 PRINT"[CD]   PRESS FIRE BUTTON TO TURN TUNE
OFF      AND ON."
20310 PRINT"[2CD]ARE YOU READY TO PLAY"
```

```
20320 GETW$:IFW$=""THEN20320
20330 IFW$="Y"THENRETURN
20340 IFW$<>"N"THEN20320
20350 PRINT"[HOME.14CD]WELL DECIDE"
20351 FORX=1TO200:NEXTX
20355 PRINT"[HOME.14CD]              "
20360 GOTO20320
25000 PRINT"[HOME,23CD,YEL]SCORE: ";S;"      LIVES
LEFT:";3-LL:RETURN
29000 RETURN
29001 POKE56333.127:POKE1.51
29002 FORI=0TO1023:POKE53248+I,PEEK(53248+I):NEXT
29003 FORI=0TO55:READA:POKE54272+I.A:NEXT
29004 POKE1.55:POKE56333.129:POKE648.196:POKE56576
.4:POKE53272.21:PRINT"[CLR]"
29005 CP=1:RETURN
30000 DATA15.62,124,248,248,252,126,15
30001 DATA240,124,62,31,31,62.124,240
30002 DATA129,195,231,255,255.126,60.24
30003 DATA24,60,126.255.255.231.195.129
30004 DATA60,126,215,215,255.255,255.173
30005 DATA24,36.90,189.189.90.36,24
30006 DATA129,189,255,215,215,255.255.173
```

### Music listings

These are given in the form of hex dumps, and should be typed in exactly as shown. The program for PESTROY, namely BOOGIETIME, comes from Chapter 13, where we look at background rhythms and music interrupt techniques: the techniques used to produce the sound in these games.

```
YELLOW SUBMARINE VOICE THREE


B*
   PC  SR AC XR YR SP
.:5F79 33 00 61 00 F6
.
.:C0B3 03 9A 04 0B 04 CF 00 00
.:C0BB 00 00 04 0B 03 9A 04 0B
.:C0C3 03 36 00 00 00 00 03 36
.:C0CB 04 0B 04 0B 03 9A 03 36
.:C0D3 02 B3 00 00 00 00 02 B3
.:C0DB 04 0B 04 0B 03 9A 00 00
```

```
.:C0E3 00 00 03 9A 04 0B 04 CF
.:C0EB 00 00 00 00 04 0B 03 9A
.:C0F3 04 0B 03 36 00 00 00 00
.:C0FB 04 0B 04 0B 03 9A 03 36
.:C103 02 B3 00 00 02 B3 04 0B
.:C10B 04 0B 03 9A 00 00 00 00
.:C113 03 9A 04 0B 04 CF 00 00
.:C11B 00 00 04 0B 03 9A 04 0B
.:C123 03 36 00 00 00 00 04 0B
.:C12B 04 0B 03 9A 03 36 02 B3
.:C133 00 00 00 00 02 B3 04 0B
.:C13B 04 0B 03 9A 00 00 00 00
.:C143 03 9A 04 0B 04 CF 00 00
.:C14B 00 00 04 0B 03 9A 04 0B
.:C153 03 36 00 00 00 00 04 0B
.:C15B 04 0B 03 9A 03 36 02 B3
.:C163 00 00 04 0B 04 0B 03 9A
.:C16B 00 00 00 00 00 00 00 00
.:C173 04 CF 04 CF 04 CF 04 CF
.:C17B 05 66 03 9A 03 9A 03 9A
.:C183 03 9A 03 9A 00 00 03 9A
.:C18B 03 9A 03 9A 03 9A 03 9A
.:C193 03 9A 00 00 03 9A 03 36
.:C19B 03 36 03 36 03 36 03 36
.:C1A3 00 00 04 CF 04 CF 04 CF
.:C1AB 04 CF 05 66 03 9A 03 9A
.:C1B3 03 9A 03 9A 03 9A 00 00
.:C1BB 03 9A 03 9A 03 9A 03 9A
.:C1C3 03 9A 03 9A 00 00 03 9A
.:C1CB 03 36 03 36 03 36 03 36
.:C1D3 03 36 00 00 04 0B 04 49
.:C1DB 04 CF 00 00 00 00 04 0B
.:C1E3 03 9A 04 0B 03 36 00 00
.:C1EB 00 00 04 0B 04 0B 03 9A
.:C1F3 03 36 02 B3 00 00 04 0B
.:C1FB 04 0B 03 9A 00 00 00 00
.:C203 03 9A 04 0B 04 CF 00 00
.:C20B 00 00 04 0B 03 9A 04 0B
.:C213 03 36 00 00 00 00 08 92
.:C21B 09 9F 0B 71 0B 71 0C D8
.:C223 0C D8 11 25 0E 6B 0B 71
.:C22B 0B 71 0E 6B 0B 71 09 9F
.:C233 0E 6B 0C D8 0A CD 0B 71
.:C23B 00 00 00 00 00 00 00 00
.:C243 00 00 00 00 FF FF FF FF
.
.
```

YELLOW SUBMARINE VOICE TWO

```
B*
   PC   SR AC XR YR SP
.:5766 33 00 4E 00 F6
.
.:C8B3 0E 6B 10 2F 13 3F 00 00
.:C8BB 00 00 10 2F 0E 6B 10 2F
.:C8C3 0C D8 00 00 00 00 0C D8
.:C8CB 10 2F 10 2F 0E 6B 0C D8
.:C8D3 0A CD 00 00 00 00 0C D8
.:C8DB 10 2F 10 2F 0E 6B 00 00
.:C8E3 00 00 0E 6B 10 2F 13 3F
.:C8EB 00 00 00 00 10 2F 0E 6B
.:C8F3 10 2F 0C D8 00 00 00 00
.:C8FB 10 2F 10 2F 0E 6B 0C D8
.:C903 0A CD 00 00 00 00 10 2F
.:C90B 10 2F 0E 6B 00 00 00 00
.:C913 0E 6B 10 2F 13 3F 00 00
.:C91B 00 00 10 2F 0E 6B 10 2F
.:C923 0C D8 00 00 00 00 10 2F
.:C92B 10 2F 0E 6B 0C D8 0A CD
.:C933 00 00 00 00 0C D8 10 2F
.:C93B 10 2F 0E 6B 00 00 00 00
.:C943 0E 6B 10 2F 13 3F 00 00
.:C94B 00 00 10 2F 0E 6B 10 2F
.:C953 0C D8 00 00 00 00 10 2F
.:C95B 10 2F 0E 6B 0C D8 0A CD
.:C963 00 00 00 00 10 2F 10 2F
.:C96B 0E 6B 00 00 00 00 00 00
.:C973 13 3F 13 3F 13 3F 13 3F
.:C97B 15 9A 0E 6B 0E 6B 0E 6B
.:C983 0E 6B 0E 6B 00 00 0E 6B
.:C98B 0E 6B 0E 6B 0E 6B 0E 6B
.:C993 0E 6B 00 00 0E 6B 0C D8
.:C99B 0C D8 0C D8 0C D8 0C D8
.:C9A3 00 00 13 3F 13 3F 13 3F
.:C9AB 13 3F 15 9A 0E 6B 0E 6B
.:C9B3 0E 6B 0E 6B 0E 6B 00 00
.:C9BB 0E 6B 0E 6B 0E 6B 0E 6B
.:C9C3 0E 6B 0E 6B 00 00 0E 6B
.:C9CB 0C D8 0C D8 0C D8 0C D8
.:C9D3 0C D8 00 00 10 2F 11 25
.:C9DB 13 3F 00 00 00 00 10 2F
.:C9E3 0E 6B 10 2F 0C D8 00 00
.:C9EB 00 00 10 2F 10 2F 0E 6B
.:C9F3 0C D8 0A CD 00 00 00 00
.:C9FB 10 2F 10 2F 0E 6B 00 00
.:CA03 10 2F 11 25 13 3F 00 00
.:CA0B 00 00 10 2F 0E 6B 10 2F
.:CA13 0C D8 00 00 00 00 00 00
```

```
.:CA1B 00 00 00 00 00 00 00 00
.:CA23 00 00 00 00 00 00 00 00
.:CA2B 00 00 00 00 00 00 00 00
.:CA33 00 00 00 00 00 00 00 00
.:CA3B 00 00 00 00 00 00 00 00
.:CA43 00 00 00 00 FF FF FF FF
.
.
```

WHEN I'M 64 VOICE THREE

```
B*
   PC  SR AC XR YR SP
.:4F53 33 00 3B 00 F6
.
.:C0B3 05 66 05 18 05 66 06 6C
.:C0BB 00 00 05 66 06 6C 07 35
.:C0C3 06 6C 08 92 00 00 00 00
.:C0CB 08 92 0A CD 08 92 07 35
.:C0D3 09 9F 00 00 08 17 08 92
.:C0DB 09 14 09 9F 08 17 08 92
.:C0E3 09 14 09 9F 08 17 07 A3
.:C0EB 00 00 07 35 00 00 08 17
.:C0F3 00 00 08 92 09 14 00 00
.:C0FB 09 9F 00 00 0A CD 0A 31
.:C103 09 9F 08 92 00 00 00 00
.:C10B 05 66 05 18 05 66 00 00
.:C113 06 6C 05 66 00 00 06 6C
.:C11B 07 35 06 6C 08 92 00 00
.:C123 0A CD 0A CD 00 00 09 9F
.:C12B 00 00 07 35 08 92 00 00
.:C133 00 00 00 00 08 92 07 35
.:C13B 08 92 00 00 0A 31 09 9F
.:C143 00 00 08 92 07 35 08 92
.:C14B 00 00 08 17 07 35 00 00
.:C153 0A CD 0A CD 00 00 0A CD
.:C15B 0A CD 00 00 08 92 00 00
.:C163 00 00 00 00 00 00 00 00
.:C16B 08 92 07 35 08 92 07 35
.:C173 08 92 07 35 08 92 07 35
.:C17B 08 92 07 35 08 92 07 35
.:C183 08 92 08 92 05 B9 06 6C
.:C18B 06 6C 06 6C 06 6C 06 6C
.:C193 06 6C 07 35 07 35 05 66
.:C19B 05 66 05 66 05 66 07 35
.:C1A3 07 35 07 35 08 17 08 17
.:C1AB 08 17 08 92 08 92 08 92
.:C1B3 0C D8 0C D8 0C D8 0C D8
.:C1BB 0C D8 0A CD 0A CD 0A CD
```

```
.:C1C3 0A CD 0A CD 0A CD 0A CD
.:C1CB 0A CD 0E 6B 0E 6B 0C D8
.:C1D3 0C D8 0A CD 0A CD 0A CD
.:C1DB 09 9F 08 92 08 92 09 9F
.:C1E3 09 9F 09 9F 09 9F 07 35
.:C1EB 07 35 07 35 07 35 08 92
.:C1F3 08 92 08 92 07 35 07 35
.:C1FB 07 35 06 6C 06 6C 06 6C
.:C203 07 35 07 35 07 35 08 92
.:C20B 08 92 00 00 FF 6B 10 2F
.
.
```

WHEN I'M 64 VOICE TWO

```
B*
   PC   SR AC XR YR SP
.:4740 33 00 28 00 F6
.
.:C8B3 05 66 05 18 05 66 06 6C
.:C8BB 00 00 05 66 06 6C 07 35
.:C8C3 06 6C 08 92 00 00 00 00
.:C8CB 08 92 0A CD 08 92 07 35
.:C8D3 09 9F 00 00 08 17 08 92
.:C8DB 09 14 09 9F 08 17 08 92
.:C8E3 09 14 09 9F 08 17 07 A3
.:C8EB 00 00 07 35 00 00 08 17
.:C8F3 00 00 08 92 09 14 00 00
.:C8FB 09 9F 00 00 0A CD 0A 31
.:C903 09 9F 08 92 00 00 00 00
.:C90B 05 66 05 18 05 66 00 00
.:C913 06 6C 05 66 00 00 06 6C
.:C91B 07 35 06 6C 08 92 00 00
.:C923 0A CD 0A CD 00 00 09 9F
.:C92B 00 00 07 35 08 92 00 00
.:C933 00 00 00 00 08 92 07 35
.:C93B 08 92 00 00 0A 31 09 9F
.:C943 00 00 08 92 07 35 08 92
.:C94B 00 00 08 17 07 35 00 00
.:C953 0A CD 0A CD 00 00 0A CD
.:C95B 0A CD 00 00 08 92 00 00
.:C963 00 00 00 00 00 00 00 00
.:C96B 08 92 07 35 08 92 07 35
.:C973 08 92 07 35 08 92 07 35
.:C97B 08 92 07 35 08 92 07 35
.:C983 08 92 08 92 05 B9 06 6C
.:C98B 06 6C 06 6C 06 6C 06 6C
.:C993 06 6C 07 35 07 35 05 66
.:C99B 05 66 05 66 05 66 07 35
.:C9A3 07 35 07 35 08 17 08 17
```

```
.:C9AB 08 17 08 92 08 92 08 92
.:C9B3 0C D8 0C D8 0C D8 0C D8
.:C9BB 0C D8 0A CD 0A CD 0A CD
.:C9C3 0A CD 0A CD 0A CD 0A CD
.:C9CB 0A CD 0E 6B 0E 6B 0C D8
.:C9D3 0C D8 0A CD 0A CD 0A CD
.:C9DB 09 9F 08 92 08 92 09 9F
.:C9E3 09 9F 09 9F 09 9F 07 35
.:C9EB 07 35 07 35 07 35 08 92
.:C9F3 08 92 08 92 07 35 07 35
.:C9FB 07 35 06 6C 06 6C 06 6C
.:CA03 07 35 07 35 07 35 08 92
.:CA0B 08 92 00 00 FF 6B 10 2F
.
.
```

# 13
# Musical Interrupts

One of the features shared by a number of popular arcade games at the moment (Frantic Freddie is a prime example) is that of musical interrupts. In other words, while the program is getting on with doing one thing, a tune miraculously carries on in the background. The purpose of this chapter is to explain, with the aid of a mixture of Basic and machine code, how it's done.

Since this whole thing is based on interrupts, there's obviously got to be something somewhere in the 64 that is capable of being interrupted. A glance at one of the many memory maps available for the machine shows that at locations 788 and 789 decimal (or $0314 and $0315 hexadecimal), there is something called the hardware interrupt vector. This usually contains the numbers 49 and 234 respectively in the two locations, or $31 and $EA in hexadecimal. This tells the 64 that every time this routine is checked, program flow must go to location $EA31, reading the numbers in low-high order as usual.

Since this routine is checked with great frequency, we can alter it (it is only RAM, after all) to jump to a routine of our own. The main thing to remember is, having made this change, to ensure that at some point in our own routine program flow still branches to that routine at $EA31, which updates the clock, amongst other things, and generally helps keep the 64 ticking over. If this call wasn't present, it would be a very sick 64 indeed, and your musical career would be at an end.

## Re-routing interrupts

So the first thing to do is alter the content of those two locations, which is really the only purpose of the first block of machine code. Set interrupt disable (in other words, we don't want to be disturbed), load the accumulator with $50 and store it at location $0314, load the accumulator with $C0 and store that at location $0315, clear interrupt disable (in other words, carry on interrupting), do nothing for a bit and return from this subroutine.

The last part of the routine sets everything back to normal.

```
MUSIC IN MACHINE CODE PART ONE

B*
   PC  SR AC XR YR SP
.:87D8 33 00 C0 00 F6
.
C000 EA          NOP
C001 EA          NOP
C002 EA          NOP
C003 78          SEI
C004 A9 50       LDA #$50
C006 8D 14 03    STA $0314
C009 A9 C0       LDA #$C0
C00B 8D 15 03    STA $0315
C00E 58          CLI
C00F EA          NOP
C010 EA          NOP
C011 60          RTS
C012 78          SEI
C013 A9 31       LDA #$31
C015 8D 14 03    STA $0314
C018 A9 EA       LDA #$EA
C01A 8D 15 03    STA $0315
C01D 58          CLI
C01E EA          NOP
C01F 60          RTS
C020 EA          NOP
C021 DE DE DE    DEC $DEDE,X
C024 DE DE DE    DEC $DEDE,X
C027 DE DE DE    DEC $DEDE,X
C02A DE DE DE    DEC $DEDE,X
C02D DE DE DE    DEC $DEDE,X
C030 DE DE DE    DEC $DEDE,X
C033 DE DE DE    DEC $DEDE,X
C036 DE DE DE    DEC $DEDE,X
C039 DE DE DE    DEC $DEDE,X
C03C DE DE DE    DEC $DEDE,X
C03F DE 0A 09    DEC $090A,X
C042 00          BRK
C043 41 00       EOR ($00,X)
C045 00          BRK
C046 00          BRK
C047 00          BRK
C048 00          BRK
C049 00          BRK
C04A 00          BRK
C04B 41 00       EOR ($00,X)
C04D 00          BRK
C04E 00          BRK
```

```
C04F 00          BRK
C050 EA          NOP
C051 EA          NOP
C052 EA          NOP
C053 AD 42 C0    LDA $C042
C056 F0 04       BEQ $C05C
C058 4C 31 EA    JMP $EA31
C05B EA          NOP
C05C CE 41 C0    DEC $C041
C05F D0 F7       BNE $C058
C061 EA          NOP
C062 EA          NOP
C063 EA          NOP
C064 AD 40 C0    LDA $C040
C067 8D 41 C0    STA $C041
C06A EA          NOP
C06B E6 FB       INC $FB
C06D D0 02       BNE $C071
C06F E6 FC       INC $FC
C071 EA          NOP
C072 A0 00       LDY #$00
C074 B1 FB       LDA ($FB),Y
C076 C9 FF       CMP #$FF
C078 F0 28       BEQ $C0A2
C07A EA          NOP
C07B EA          NOP
C07C EA          NOP
C07D EA          NOP
C07E A0 00       LDY #$00
C080 8D 0F D4    STA $D40F
C083 E6 FB       INC $FB
C085 D0 02       BNE $C089
C087 E6 FC       INC $FC
C089 EA          NOP
C08A B1 FB       LDA ($FB),Y
C08C 8D 0E D4    STA $D40E
C08F AD 43 C0    LDA $C043
C092 48          PHA
C093 A9 00       LDA #$00
C095 8D 12 D4    STA $D412
C098 68          PLA
C099 8D 12 D4    STA $D412
C09C EA          NOP
C09D EA          NOP
C09E 4C 64 C8    JMP $C864
C0A1 EA          NOP
C0A2 EA          NOP
C0A3 EA          NOP
C0A4 A5 FD       LDA $FD
C0A6 85 FB       STA $FB
C0A8 A5 FE       LDA $FE
C0AA 85 FC       STA $FC
```

```
C0AC 4C 64 C0    JMP $C064
C0AF 00          BRK
C0B0 00          BRK
C0B1 00          BRK
C0B2 00          BRK
.
.
```

MUSIC IN MACHINE CODE PART TWO

```
B*
   PC  SR AC XR YR SP
.:77B2 33 00 9A 00 F6
.
C800 EA          NOP
C801 EA          NOP
C802 EA          NOP
C803 78          SEI
C804 A9 50       LDA #$50
C806 8D 14 03    STA $0314
C809 A9 C8       LDA #$C8
C80B 8D 15 03    STA $0315
C80E 58          CLI
C80F EA          NOP
C810 EA          NOP
C811 60          RTS
C812 78          SEI
C813 A9 31       LDA #$31
C815 8D 14 03    STA $0314
C818 A9 EA       LDA #$EA
C81A 8D 15 03    STA $0315
C81D 58          CLI
C81E EA          NOP
C81F 60          RTS
C820 EA          NOP
C821 DE DE DE    DEC $DEDE,X
C824 DE DE DE    DEC $DEDE,X
C827 DE DE DE    DEC $DEDE,X
C82A DE DE DE    DEC $DEDE.X
C82D DE DE DE    DEC $DEDE,X
C830 DE DE DE    DEC $DEDE,X
C833 DE DE DE    DEC $DEDE,X
C836 DE DE DE    DEC $DEDE,X
C839 DE DE DE    DEC $DEDE,X
C83C DE DE DE    DEC $DEDE,X
C83F DE 0A 0A    DEC $0A0A,X
C842 00          BRK
C843 41 00       EOR ($00,X)
C845 00          BRK
```

```
C846 00          BRK
C847 00          BRK
C848 00          BRK
C849 00          BRK
C84A 00          BRK
C84B 41 00       EOR ($00,X)
C84D 00          BRK
C84E 00          BRK
C84F 00          BRK
C850 EA          NOP
C851 EA          NOP
C852 EA          NOP
C853 AD 42 C8    LDA $C842
C856 F0 04       BEQ $C85C
C858 4C 31 EA    JMP $EA31
C85B EA          NOP
C85C CE 41 C8    DEC $C841
C85F D0 F7       BNE $C858
C861 EA          NOP
C862 EA          NOP
C863 EA          NOP
C864 AD 40 C8    LDA $C840
C867 8D 41 C8    STA $C841
C86A EA          NOP
C86B E6 F7       INC $F7
C86D D0 02       BNE $C871
C86F E6 F8       INC $F8
C871 EA          NOP
C872 A0 00       LDY #$00
C874 B1 F7       LDA ($F7),Y
C876 C9 FF       CMP #$FF
C878 F0 28       BEQ $C8A2
C87A EA          NOP
C87B EA          NOP
C87C EA          NOP
C87D EA          NOP
C87E A0 00       LDY #$00
C880 8D 08 D4    STA $D408
C883 E6 F7       INC $F7
C885 D0 02       BNE $C889
C887 E6 F8       INC $F8
C889 EA          NOP
C88A B1 F7       LDA ($F7),Y
C88C 8D 07 D4    STA $D407
C88F AD 43 C8    LDA $C843
C892 48          PHA
C893 A9 00       LDA #$00
C895 8D 0B D4    STA $D40B
C898 68          PLA
C899 8D 0B D4    STA $D40B
C89C EA          NOP
C89D EA          NOP
```

```
C89E 4C 31 EA     JMP $EA31
C8A1 EA           NOP
C8A2 EA           NOP
C8A3 EA           NOP
C8A4 A5 F9        LDA $F9
C8A6 85 F7        STA $F7
C8A8 A5 FA        LDA $FA
C8AA 85 F8        STA $F8
C8AC 4C 64 C8     JMP $C864
C8AF 00           BRK
C8B0 00           BRK
C8B1 00           BRK
C8B2 00           BRK
.
.
```

What have we achieved by doing this? Well, program flow from the hardware interrupt vector no longer goes off to location $EA31 directly, it now goes there via location $C050 instead. This is set up by typing in SYS49152, and everything is returned to normal by typing SYS49171. However, since we haven't got any code at location $C050, there's not a lot of point in typing this just yet.

## Getting going

Without an assembler, typing all this in is going to be tricky from the machine code listings, and so the data program has been included as well. This is in fact longer than the machine code equivalent, since the listings only show the relevant parts of the machine code, not the gaps in between. But if you haven't got an assembler you'll just have to resign yourself to doing a bit more typing.

```
BOOGIEBASIC PART1


10 FORI=0TO71:READA:B=B+A:POKE49152+I.A:NEXT
12 IFB<>11101THENPRINT"ERROR IN BLOCK #0":STOP
20 FORI=72TO143:READA:C=C+A:POKE49152+I.A:NEXT
22 IFC<>11286THENPRINT"ERROR IN BLOCK #1":STOP
30 FORI=144TO175:READA:D=D+A:POKE49152+I.A:NEXT
32 IFD<>4866THENPRINT"ERROR IN BLOCK #2":STOP
34 PRINT"PART ONE OF DATA ENTERED CORRECTLY AND  N
OW IN PLACE.":END
30000 REM BLOCK # 0
30001 DATA234,234,234,120,169,80,141,20
30002 DATA3,169,192,141,21,3,88,234
```

```
30003 DATA234,96,120,169,49,141,20,3
30004 DATA169.234,141.21,3,88,234,96
30005 DATA234,222.222,222,222,222,222,222
30006 DATA222.222,222.222,222,222.222,222
30007 DATA222,222,222,222,222,222,222,222
30008 DATA222.222,222,222,222,222,222,222
30009 DATA10,9,0,65,0,0,0,0
30010 REM BLOCK # 1
30011 DATA0,0,0,65,0,0,0,0
30012 DATA234.234,234,173.66,192,240,4
30013 DATA76,49,234,234,206,65,192,208
30014 DATA247.234.234.234,173,64,192,141
30015 DATA65,192,234,230,251,208,2.230
30016 DATA252.234,160.0,177,251,201,255
30017 DATA240,40,234,234,234,234,160,0
30018 DATA141.15,212,230,251,208,2,230
30019 DATA252,234,177,251,141,14,212,173
30020 REM BLOCK # 2
30021 DATA67,192,72,169,0,141,18,212
30022 DATA104.141,18,212,234,234,76,100
30023 DATA200,234,234,234,165,253,133,251
30024 DATA165.254,133,252,76,100,192,0

READY.



BOOGIEBASIC PART2


10 FORI=0TO71:READA:B=B+A:POKE51200+I.A:NEXT
12 IFB<>11110THENPRINT"ERROR IN BLOCK #0":STOP
20 FORI=72TO143:READA:C=C+A:POKE51200+I.A:NEXT
22 IFC<>11280THENPRINT"ERROR IN BLOCK #1":STOP
30 FORI=144TO175:READA:D=D+A:POKE51200+I.A:NEXT
32 IFD<>4835THENPRINT"ERROR IN BLOCK #2":STOP
40 PRINT"PART TWO OF DATA ENTERED CORRECTLY AND  N
OW IN PLACE.":END
30000 REM BLOCK # 0
30001 DATA234,234,234,120.169.80,141.20
30002 DATA3,169,200,141,21,3,88,234
30003 DATA234,96,120,169,49,141,20,3
30004 DATA169,234,141,21,3.88,234,96
30005 DATA234,222,222,222,222,222,222,222
30006 DATA222,222,222.222,222,222,222,222
30007 DATA222,222,222,222,222,222,222.222
30008 DATA222.222,222.222,222,222,222,222
30009 DATA10,10,0,65,0,0,0,0
30010 REM BLOCK # 1
30011 DATA0,0,0,65,0,0,0,0
30012 DATA234.234.234,173,66,200,240.4
```

```
30013 DATA76,49,234,234,206,65,200,208
30014 DATA247,234,234,234,173,64,200,141
30015 DATA65,200,234,230,247,208,2,230
30016 DATA248,234,160,0,177,247,201,255
30017 DATA240,40,234,234,234,234,160,0
30018 DATA141,8,212,230,247,208,2,230
30019 DATA248,234,177,247,141,7,212,173
30020 REM BLOCK # 2
30021 DATA67,200,72,169,0,141,11,212
30022 DATA104,141,11,212,234,234,76,49
30023 DATA234,234,234,234,165,249,133,247
30024 DATA165,250,133,248,76,100,200,0
```

The last two blocks of machine code (the hexadecimal dumps) are the actual tune that is going to be played: in this case, a simple boogie riff, since my musical talent is minimal. Again, these can be typed in either with an assembler, or you'll have to do it the long way and work out by hand where all the code lies.

```
B*
   PC  SR AC XR YR SP
.:6F9F 33 00 87 00 F6
.
.:C0B3 04 49 00 00 04 49 00 00
.:C0BB 04 49 00 00 04 49 00 00
.:C0C3 04 49 00 00 04 49 00 00
.:C0CB 04 49 00 00 04 49 00 00
.:C0D3 05 B9 00 00 05 B9 00 00
.:C0DB 05 B9 00 00 05 B9 00 00
.:C0E3 04 49 00 00 04 49 00 00
.:C0EB 04 49 00 00 04 49 00 00
.:C0F3 06 6A 00 00 06 6A 00 00
.:C0FB 05 B9 00 00 05 B9 00 00
.:C103 FF FF FF FF FF FF FF FF
.
.
```

```
B*
   PC  SR AC XR YR SP
.:678C 33 00 74 00 F6
.
.:C8B3 04 49 05 67 06 6A 07 35
.:C8BB 07 A3 07 35 06 6A 05 67
```

```
.:C8C3 04 49 05 67 06 6A 07 35
.:C8CB 07 A3 07 35 06 6A 05 67
.:C8D3 05 B9 07 45 08 93 09 9F
.:C8DB 0A 3C 09 9F 08 93 07 45
.:C8E3 04 49 05 67 06 6A 07 35
.:C8EB 07 A3 07 35 06 6A 05 67
.:C8F3 06 6A 08 17 09 9F 08 17
.:C8FB 05 B9 07 35 08 93 07 35
.:C903 FF DF FF FF FF FF FF FF
.
.
```

The final bit of typing is the Basic program headed Music Driver Program. Straightforward, since it's all in Basic. The only line needing an explanation at the moment, in order that you can get it working before we dive into the great explanation of how it all does what it does, is line 5. This checks a memory location where part of the machine code lies, and if that location doesn't contain a 65, then we load the program 'BOOGIETIME' from disk. If you're using a tape system, change that eight to a one. 'Boogietime' is just the name I gave to the whole of the machine code collection of routines.

```
MUSIC DRIVER PROGRAM


10 S=54272:POKES+17,255:POKES+19,142:POKES+20,150
20 POKES+24,15:POKE49216,10:POKE49218,0:POKE49219,
65:POKE251,178:POKE252,192
30 POKE253,178:POKE254,192
100 POKES+10,255:POKES+12,142:POKES+13,150
110 POKE51264,10:POKE51266,0:POKE51267,65:POKE247,
178:POKE248,200
120 POKE249,178:POKE250,200:SYS49152
```

If you have got it all typed in, just run the Basic program as a normal Basic program, and sit back and listen to the music. Now, the great question, how does it all work?

### Basic program explanation

Taking this line by line, we have:

Line 10 : Set the variable S to equal the start of the SID chip, and set the low order pulse frequency for voice 3 to be maximum, give it an attack/decay setting of 142 and a sustain/release setting of 150.

Line 20 : Turn the volume up to maximum, and POKE some values which the machine code program will later use.

Line 30 : More values for the machine code program.

Lines 100-120 : Do the same for voice 2 before activating the routine with a call of SYS49152.

Before running this program, do make sure that the machine code routines are resident in memory. Load the machine code programs with a 'LOAD "NAME",8,1', perform a CLR and NEW, load (or type in) the Basic program, and then RUN it.

**Machine code explanation**

We'll just concentrate on the routine for voice 3 (which lies from locations $C050 to $C0AF), as the one for voice 2 is virtually identical.

First of all, check the content of location $C042 (49218), and if it contains a zero then carry on with the program. If not, jump to the internal routine at $EA31 and carry on as normal.

Decrease the content of location $C041 (49217) and if that isn't equal to zero, branch back to $C058 and off to $EA31 again.

Load the accumulator with the content of location $C040 and store it at $C041. These are all the locations that our earlier Basic program poked when it was setting everything up.

Now the next lot of instructions, concerning locations $FB, $FC and so on, requires a bit of discussion. The locations used (and the ones in block four of the machine code as well) form part of what is referred to as page zero of the 64's memory map. It so happens that these locations don't do very much, so we can use them ourselves to store numbers in. The Basic program earlier poked some values into these locations, 178 and 192 for locations $FB and $FC respectively, which form, when linked together, the hexadecimal number $C0B2 (since 178 is $B2 in hexadecimal, 192 is $C0, and we reverse the order of the numbers as usual).

What the rest of this machine code program is doing is to load the accumulator with the value to be found in locations $C020 onwards, and store that value in the locations that form the high and low order frequencies for the note to be played by voice 3.

Complicated, eh? This 'offset' machine code instruction is not one of the easiest in the world to understand, and many competent machine code programmers will probably never use it, which is a shame, since it makes this type of program so much easier to write. Without using it, the code needed here would be about five times as long as is necessary.

The latter part of the program turns the waveform off and on again (as should always be done before playing a note, otherwise you'll usually just get silence), jumps to the other routine to play the next note for voice 2, and then finally sets everything back to normal and goes back to start all over again.

The end of a tune is indicated by a value $FF, or decimal 255, being read, so just make sure that none of your musical notes contain this value.

By altering what is stored in locations $C0B2 and $C8B2 onwards, you can change the tune that is being played. Since this code is well spaced out, there is room to have something like 900 notes in each tune before running out of space, which should be enough to keep most people happy.

## Conclusion

You'll probably only really understand what's going on by typing everything in, getting it running, and then experimenting with it to see what happens. Do make sure that you save copies of everything to tape or disk before attempting to run it, because a simple error in entering the code can so easily cause the machine to crash.

By changing the ADSR settings for the voices, perhaps experimenting with different wave forms, different pulse widths, and so on, you should be able to produce some excellent background music for your programs. Voice one is still available to perform as normal, so you can still have various sound effects being played as well as the tune.

Have fun!

# 14
# Adding Musical Commands to Basic

## Introduction

In this chapter we'll be looking at some of the ways in which you can enhance the existing Basic in your Commodore 64. Although the methods used in this chapter are by no means the only ways in which this can be done, they at least have the virtue of working!

A number of routines are also presented here as well, and these could readily be incorporated into one complete package, instead of consisting of a lot of separate little programs as they do now.

Having read this chapter you should be in a position to devise some new commands of your own, and insert these into whatever spare memory space you see fit. Here most of the routines have been written to start at location 49152 (or $C000), since this is a conveniently empty block of memory on the Commodore 64 at power on.

The commands are presented in the form of machine code disassemblies, in order to give you a better chance of understanding how they work, and occasionally Basic loader programs for those of you unfortunate enough not to possess an assembler.

If you've got the tape that accompanies this book, then there won't be any problems when it comes to entering the code. However, if your wallet or fingers aren't up to it, then you can still use the commands by converting the numbers as shown in the listings, and hand POKEing them into the machine.

Either way, you'll still be able to use them. However, an assembler/disassembler is a powerful tool for the machine code programmer, whether beginner or advanced, and I strongly recommend that one way or another you get a program of that sort.

For now, let's take a look at what Commodore Basic has and hasn't got.

## Commodore's Basic

The deficiencies inherent in Commodore Basic are well known. But it's interesting to trace these deficiencies back through time to the very early Commodore machines.

The first Commodore PET, as well as coming complete with its own cassette deck and monitor, and having a paltry 8K of RAM (and also costing some £625 when it first appeared back in 1979!), had what Commodore themselves termed Basic 1.

As a Basic language it was fine at the time, but there were a number of things missing from it. For example, there was no way of accessing the machine code monitor, as it didn't have one built in.

A utility to overcome this soon came on the market, but this took up precious space from the meagre amount of RAM that was there to start with, and so Commodore followers had to wait a couple of years before Basic 2 appeared.

### Basic 2

When Basic 2 did appear it caused instant confusion among the Commodore ranks, since some people were calling it Basic 2, and others referred to it as Basic 3. Seemingly the so-called 'Basic 2' never appeared, and although this particular version of the language was always called Basic 2, theoretically it should have been referred to as Basic 3.

Whatever number you gave it, it was a great improvement over its predecessor, and did have access to a machine code monitor. Moreover, the ROM installed was now capable of looking after disk drives, something that the earlier machines could not do.

Time went by, Basic 4 appeared, and for a long time it was rumoured that there was to be a fifth version of the language as well, with all the features that existing ones had lacked, of which more in a moment.

However, at the time of writing Basic 5 (or something closely resembling it) is only available on the new Plus Four and Commodore 16,

so for us 64 owners ...

The version of Basic that they've installed in the 64 is that which we referred to earlier as Basic 2/3, although Commodore have apparently now decided that it should be called Basic 2, and indeed this is what the machine greets you with when you turn it on.

However, not only have Commodore taken a retrograde step and installed an old version of their popular Basic language, but they've also managed to take out a great deal of what was already in there. So we see no machine code monitor - hence the need for programs such as Extramon.

What we are left with instead is an extremely flexible memory management system, but a very poor Basic with which to manage it. To look after all this requires a lot of work, and to understand it all properly requires even more!

### Basic advantages

The version of Basic in the Commodore 64 is a pretty standard version of what is usually referred to as Microsoft Basic.

This is based on the original Beginners All-purpose Symbolic Instruction Code, from which Basic takes its name. This language was devised a number of years ago, and the cracks are now beginning to show, but for a beginner it is still possibly the easiest of languages to learn.

Apart from the interface to machine code, which is not good, the commands you have at your disposal are not too difficult to understand. Because of the great similarity between Basic words and English words, most beginners can soon start writing programs in Basic.

### And disadvantages

However, most beginners also soon come to realise that the version of Basic as supplied by Commodore is sadly lacking in a number of departments.

The concepts of structured programming, the computer flavour of the month, are impossible to simulate on the 64, and there is a distinct lack of such commands as PRINT AT, PRINT USING, and so on.

In particular, when it comes to using graphics and sound, the number of commands is strictly limited to two : PEEK and POKE. No other commands exist to cope with the vast number of PEEKs and POKEs needed to set up a high resolution screen and draw things on it, or to play a few musical notes, or do just about anything with either graphics or sound.

If you want to make music, or display various images on the screen, it has all got to be done the long way, by using a laborious series of POKEs. Given that this version of Basic is so appalling in these particular departments, it is no wonder that people go to great lengths to try to improve it. There are now many packages on the market that have set out to try to improve on the language that we are originally offered, in a variety of different ways.

Whether they succeed in their chosen aims is, of course, a completely different matter, but what they all have in common is that they are adding commands to the existing version of Basic, and through those commands are seeking to make life easier for the person using the machine.

The rest of this chapter will be devoted to showing you two ways in which commands could be added, as well as giving you a number of routines to try out for yourself.

But first, the concepts involved.

## Adding commands: the concepts

There are many different ways in which you can add commands to Commodore's existing command set. Commands can be added either as words or symbols, or indeed we could also use the function keys: re-define them to be able to accept existing Basic keywords, and then put our new words (or symbols) in their place.

We'll be looking at the two simplest options in this chapter, namely defining various symbols to act as commands, rather than adding new words, and using a direct SYStem call to interpret a new command.

These are certainly easier than trying to add new command words to Basic, as this involves altering a lot more things than we are going to do, and for the first-time user can seem incredibly complicated. So complicated in fact that you probably wouldn't even want to try it!

Still, what we are going to do is fairly straightforward, and shouldn't present any major difficulties.

### Getting a character

Anything that you type on to the screen is interpreted and executed by the Commodore 64 as soon as you press the return key. Once this key has been pressed there are a number of routines built into the 64 which will act upon everything that you typed in, and depending on precisely what you typed a number of things will happen.

You can generate a syntax error, and a subroutine exists within the Basic ROM to print out a suitable message and return to await your next input. Since it is in ROM (it starts at location $AF08) we can't alter it, but there's nothing to stop us copying this ROM into RAM and altering it there, so that SYNTAX ERROR becomes something a lot more meaningful. Or a lot more rude, if you're feeling in that kind of mood!

You could have entered a line of a program, in which case you won't get any error messages (or for that matter any other messages) coming back at all, but a great many pointers inside the machine will have been altered to cope with the new line.

You might have entered a direct command, and in this case the machine will just execute whatever it was that you typed in.

### Character get routine

How does the machine know what to do? In other words, how does it interpret what you've typed in? Understanding this is the key to generating our own commands, because if we can intercept the Basic routine that looks after all the commands and alter it, we are then well on the way to adding our own commands into the machine.

The machine knows what to do because of the ROM that's built into it, but there must be a routine somewhere in the machine that looks at what you've typed in and thinks 'ahah!', and then does (or attempts to do) whatever you've told it.

There is indeed such a routine, which lives in locations $0073 to $008A (or decimal locations 115 to 138), and this is usually referred to as the CHARGET routine, or character get.

This routine gets a character that you've typed in and acts upon that character.

The routine looks, in its original form, like this:

```
CHARACTER GET ROUTINE BEFORE


B*
   PC  SR AC XR YR SP
.:8FEB 33 00 D3 00 F6
.
0073 E6 7A        INC $7A
0075 D0 02        BNE $0079
0077 E6 7B        INC $7B
0079 AD 31 02     LDA $0231
007C C9 3A        CMP #$3A
007E B0 0A        BCS $008A
0080 C9 20        CMP #$20
0082 F0 EF        BEQ $0073
0084 38           SEC
0085 E9 30        SBC #$30
0087 38           SEC
0088 E9 D0        SBC #$D0
008A 60           RTS
.
.
```

What we are going to do is alter that routine so that it no longer behaves in quite the same way.

As it stands at the moment, it interprets everything in the following way:

| | | |
|---|---|---|
| Locations $73-$77 | : | update the pointer in memory locations $7A and $7B. |
| Locations $79-$7B | : | this is the pointer. |
| Locations $7C-$7F | : | if it's a colon or greater, then end. |
| Locations $80-$83 | : | if it's a space, then loop back to start again. |
| Locations $84-$8A | : | set flags for character type, and return from subroutine. |

**Comments**

This routine is the key to adding commands to Basic, since by altering it we can make it jump to some code of our own which will check for a special character, and if that character has been entered then do something! If we find that a special character has not been typed, then it's back to the routine again and carry on as normal.

We'll see later on how we can actually load a program into the computer which, when executed, alters the routine to behave in the way we want.

Instead, a couple of JSRs (jumps to subroutines) will be incorporated in it, and when we've finished with it it will look like this:

```
AND AFTER!


B*
   PC  SR AC XR YR SP
.;7FC5 33 00 AD 00 F6
.
0073 E6 7A        INC $7A
0075 D0 02        BNE $0079
0077 E6 7B        INC $7B
0079 AD 1E 02     LDA $021E
007C C9 3A        CMP #$3A
007E F0 0A        BEQ $008A
0080 C9 20        CMP #$20
0082 F0 EF        BEQ $0073
0084 20 00 C2     JSR $C200
0087 20 00 C1     JSR $C100
008A 60           RTS
.
.
```

It would be wise, at this point, to make an effort to get an assembler up and loaded into the computer, since this will make life a lot easier from now on. Without it we can still proceed with a lot of POKEs, but in order to see precisely what is happening, an assembler is a great help.

## Altering the CHARGET routine

When everything is running normally, on pressing the Return key the system will come out of ROM into this routine to fetch the next character of Basic text, then trundle back into ROM again to ponder on its next move.

What will happen now is that the system will come out of ROM, to our changed subroutine, and when it hits the first JSR command it will jump to the routine sitting at location $C200 onwards. This will determine whether or not we're going to be interpreting a special command, and if we are jump somewhere else to process it.

If we're not, then the system goes back to the altered CHARGET routine, and finds that it now has to make yet another jump, this time to location $C100. This is simply a direct copy of what used to exist in the portion of CHARGET that we have changed, so that execution can continue as normal in the event of a special character not being found.

After that, it returns into ROM again to work out what will happen next.

The program to alter the CHARGET routine sits at locations $C10B onwards, and together with the direct replacement for the altered parts, which starts at location $C100, it all looks like this :

```
B*
   PC  SR AC XR YR SP
.:8FEB 33 00 D3 00 F6
.
C100 C9 3A        CMP #$3A
C102 B0 06        BCS $C10A
C104 38           SEC
C105 E9 30        SBC #$30
C107 38           SEC
C108 E9 D0        SBC #$D0
C10A 60           RTS
C10B A9 20        LDA #$20
C10D 85 84        STA $84
```

```
C10F 85 87        STA $87
C111 A9 00        LDA #$00
C113 85 85        STA $85
C115 85 88        STA $88
C117 A9 C2        LDA #$C2
C119 85 86        STA $86
C11B A9 C1        LDA #$C1
C11D 85 89        STA $89
C11F A9 F0        LDA #$F0
C121 85 7E        STA $7E
C123 A9 04        LDA #$04
C125 85 7A        STA $7A
C127 85 7B        STA $7B
.
.
```

The next routine that we need is the one to separate the extra code and the processing of that code from ordinary Basic. In this routine we check the current character being processed against a table stored at locations $C300 onwards, and if we find what we're looking for, branch to the appropriate subroutine by reading the most significant byte and least significant byte of the subroutine address from a table which is stored immediately after the character data.

```
B*
   PC   SR AC XR YR SP
.;371A 33 00 02 00 F6
.
C200 08           PHP
C201 86 04        STX $04
C203 A2 04        LDX #$04
C205 DD 00 C3     CMP $C300,X
C208 F0 07        BEQ $C211
C20A CA           DEX
C20B 10 F8        BPL $C205
C20D A6 04        LDX $04
C20F 28           PLP
C210 60           RTS
C211 BD 06 C3     LDA $C306,X
C214 8D 1E C2     STA $C21E
C217 BD 08 C3     LDA $C308,X
C21A 8D 1F C2     STA $C21F
C21D 20 00 C0     JSR $C000
C220 20 74 A4     JSR $A474
.
.
```

To explain what's happening, the program first of all saves the cur-

rent status register on to the stack, and the current value held in the X register into location $0004 in the event of not finding a special character.

If that is the case, then everything is read back into the appropriate registers and it's off to the CHARGET routine again.

### Finding special characters

However, if a special character is found then we branch out to location $C211 where we get the least significant byte and the most significant byte from our table. These are then stored at the appropriate registers, and then the program branches off to the subroutine to carry out the command.

The characters and their LSBs and MSBs look like this:

```
B*
   PC   SR AC XR YR SP
.;26F4 33 00 DC 00 F6
.
C300 5F           ???
C301 21 21        AND ($21,X)
C303 21 21        AND ($21,X)
C305 00           BRK
C306 00           BRK
C307 C0 C0        CPY #$C0
.
.
```

This may not look very sensible as a disassembly, but it's the data that we're after, not the annotations.

The only thing we need know is a routine to execute, and in this case we've used an OLD routine. This can be used without the rest of this code by just typing in SYS49152, and it will then recover any program lost after a NEW command had been issued.

On the other hand, there's something infinitely more satisfying about seeing your own code being executed at the press of a key, rather than typing in boring old SYS commands all the time.

```
B*
   PC  SR AC XR YR SP
.;6F9F 33 00 87 00 F6
.
C000 A5 2B       LDA $2B
C002 A4 2C       LDY $2C
C004 85 22       STA $22
C006 84 23       STY $23
C008 A0 03       LDY #$03
C00A C8          INY
C00B B1 22       LDA ($22),Y
C00D D0 FB       BNE $C00A
C00F C8          INY
C010 98          TYA
C011 18          CLC
C012 65 22       ADC $22
C014 A0 00       LDY #$00
C016 91 2B       STA ($2B),Y
C018 A5 23       LDA $23
C01A 69 00       ADC #$00
C01C C8          INY
C01D 91 2B       STA ($2B),Y
C01F 88          DEY
C020 A2 03       LDX #$03
C022 E6 22       INC $22
C024 D0 02       BNE $C028
C026 E6 23       INC $23
C028 B1 22       LDA ($22),Y
C02A D0 F4       BNE $C020
C02C CA          DEX
C02D D0 F3       BNE $C022
C02F A5 22       LDA $22
C031 69 02       ADC #$02
C033 85 2D       STA $2D
C035 A5 23       LDA $23
C037 69 00       ADC #$00
C039 85 2E       STA $2E
C03B 60          RTS
.
.
```

Now that we've got everything together, it only remains to run the program by going to the various parts of it, and then, just by pressing the left arrow key and return, we can instantly recover any program that may have been lost due to an accidental NEW.

To add yet more commands, you'll need to store more data for characters, and more data for LSBs and MSBs at locations $C3000 onwards (or anywhere else for that matter, as long as the program is pointed

to the correct location!). The data for the characters is just the ASCII code for each character.

If, however, your forte is typing words rather than symbols, the following program shows how you might use the word BAK to get back a program, rather than typing in the left-arrow key.

```
B*
   PC  SR AC XR YR SP
.;8D55 31 72 9F 00 F6
.
C200 08          PHP
C201 86 04       STX $04
C203 A2 00       LDX #$00
C205 DD 00 C3    CMP $C300,X
C208 F0 26       BEQ $C230
C20A CA          DEX
C20B 10 F8       BPL $C205
C20D A6 04       LDX $04
C20F 28          PLP
C210 60          RTS
C211 BD 06 C3    LDA $C306,X
C214 8D 1E C2    STA $C21E
C217 BD 08 C3    LDA $C308,X
C21A 8D 1F C2    STA $C21F
C21D 20 C0 FF    JSR $FFC0
C220 E6 C9       INC $C9
C222 D0 02       BNE $C226
C224 E6 CA       INC $CA
C226 28          PLP
C227 A2 00       LDX #$00
C229 A1 C9       LDA ($C9,X)
C22B A6 04       LDX $04
C22D 60          RTS
C22E 00          BRK
C22F 00          BRK
C230 E6 7A       INC $7A
C232 D0 02       BNE $C236
C234 E6 7B       INC $7B
C236 A2 00       LDX #$00
C238 A1 7A       LDA ($7A,X)
C23A 38          SEC
C23B E9 41       SBC #$41
C23D F0 03       BEQ $C242
C23F 4C 5A C2    JMP $C25A
C242 E6 7A       INC $7A
C244 D0 02       BNE $C248
C246 E6 7B       INC $7B
C248 A2 00       LDX #$00
C24A A1 7A       LDA ($7A,X)
```

```
C24C 38          SEC
C24D E9 4B       SBC #$4B
C24F F0 03       BEQ $C254
C251 4C 08 AF    JMP $AF08
C254 20 00 C0    JSR $C000
C257 20 74 A4    JSR $A474
```

This, however, is not the only way of adding commands to Basic. We mentioned earlier the possibilities of doing a direct SYStem call, and that is indeed what we will look at next.

## System calling

This method of adding commands arose from an article in *Commodore Computing International* magazine, published in October 1983. The routine in there was to aid in designing your own characters, and the idea behind it was so interesting that it had to be pursued further. For the benefit of anyone who didn't see that original article, here is the listing that started it all off:

```
10 FORI=0TO71:READA:B=B+A:POKE49152+I,A:NEXT
12 IFB<>8515THENPRINT"ERROR IN BLOCK #0":STOP
20 FORI=72TO143:READA:C=C+A:POKE49152+I,A:NEXT
22 IFC<>7732THENPRINT"ERROR IN BLOCK #1":STOP
30 FORI=144TO215:READA:D=D+A:POKE49152+I,A:NEXT
32 IFD<>10278THENPRINT"ERROR IN BLOCK #2":STOP
40 FORI=216TO287:READA:E=E+A:POKE49152+I,A:NEXT
42 IFE<>5944THENPRINT"ERROR IN BLOCK #3":STOP
50 FORI=288TO335:READA:F=F+A:POKE49152+I,A:NEXT
52 IFF<>5834THENPRINT"ERROR IN BLOCK #4":STOP
60 PRINT"DATA ENTERED CORRECTLY AND CODE NOW IN  P
LACE.":END
30000 REM BLOCK # 0
30001 DATA169,0,133,87,169,208,133,88
30002 DATA169,0,133,89,169,176,133,90
30003 DATA173,14,220,41,254,141,14,220
30004 DATA165,1,41,251,133,1,160,0
30005 DATA177,87,145,89,24,165,87,105
30006 DATA1,133,87,133,89,144,241,230
30007 DATA88,230,90,165,88,201,224,208
30008 DATA231,165,1,9,4,133,1,173
30009 DATA14,220,9,1,141,14,220,173
30010 REM BLOCK # 1
30011 DATA2,221,9,3,141,2,221,173
```

```
30012 DATA0,221,41,252,9,1,141,0
30013 DATA221,169.128,141,136,2.133,56
30014 DATA133,52,169,12,141,24,208,32
30015 DATA68.229,96,0,0,0,0,0
30016 DATA0,0,0,0,0,0,0,0
30017 DATA0,0,0,0,127,0,33,0
30018 DATA160,255,255,255,255,255,255,255
30019 DATA255.255,255,255,255,255,255,255
30020 REM BLOCK # 2
30021 DATA255,255,255.255,255,255.255,255
30022 DATA255,255,255,255,255,255,255,255
30023 DATA0,0,0,32,0,0,0,0
30024 DATA0,0,0,0,0,0,0,242
30025 DATA0,0,0,0,0,0,0,0
30026 DATA0,0,0,0,0,0,0,0
30027 DATA255,127,255,255,255,255,255,247
30028 DATA251,255,255,255,239,251,255,223
30029 DATA255,255,251,255,255,255,255.255
30030 REM BLOCK # 3
30031 DATA255,255,255,255,255,255.251,255
30032 DATA0,0,8,0,0,0,0,0
30033 DATA0,0,0,0,0,0,0,0
30034 DATA0,0,0,0,0,0,0,0
30035 DATA0,0,0,0,127,0,33,0
30036 DATA32,253,174,32,235,183,132,91
30037 DATA133,92,160,3,165,20,133,88
30038 DATA165,21,133,89,134,90,24,6
30039 DATA88,38,89,136,208,248,169,176
30040 REM BLOCK # 4
30041 DATA24,101,89,133,89,201,191,144
30042 DATA3,76,72,178,169,0,133,87
30043 DATA165,90,164,87,145,88,162,0
30044 DATA164,91,165,92,32,241,183,134
30045 DATA90,230,87,165,87,201,7,208
30046 DATA231,165,90,164,87,145,88,96
```

although it didn't look quite like this in the magazine. The idea is fairly simple. There are two machine code routines, one starting at location $C000, and one at location $C100.

```
B*
   PC  BR AC XR YR SP
.:97FE 33 00 E6 00 F6
.
C000 A9 00      LDA #$00
C002 85 57      STA $57
C004 A9 D0      LDA #$D0
C006 85 58      STA $58
```

```
C008 A9 00      LDA #$00
C00A 85 59      STA $59
C00C A9 B0      LDA #$B0
C00E 85 5A      STA $5A
C010 AD 0E DC   LDA $DC0E
C013 29 FE      AND #$FE
C015 8D 0E DC   STA $DC0E
C018 A5 01      LDA $01
C01A 29 FB      AND #$FB
C01C 85 01      STA $01
C01E A0 00      LDY #$00
C020 B1 57      LDA ($57),Y
C022 91 59      STA ($59),Y
C024 18         CLC
C025 A5 57      LDA $57
C027 69 01      ADC #$01
C029 85 57      STA $57
C02B 85 59      STA $59
C02D 90 F1      BCC $C020
C02F E6 58      INC $58
C031 E6 5A      INC $5A
C033 A5 58      LDA $58
C035 C9 E0      CMP #$E0
C037 D0 E7      BNE $C020
C039 A5 01      LDA $01
C03B 09 04      ORA #$04
C03D 85 01      STA $01
C03F AD 0E DC   LDA $DC0E
C042 09 01      ORA #$01
C044 8D 0E DC   STA $DC0E
C047 AD 02 DD   LDA $DD02
C04A 09 03      ORA #$03
C04C 8D 02 DD   STA $DD02
C04F AD 00 DD   LDA $DD00
C052 29 FC      AND #$FC
C054 09 01      ORA #$01
C056 8D 00 DD   STA $DD00
C059 A9 80      LDA #$80
C05B 8D 88 02   STA $0288
C05E 85 38      STA $38
C060 85 34      STA $34
C062 A9 0C      LDA #$0C
C064 8D 18 D0   STA $D018
C067 20 44 E5   JSR $E544
C06A 60         RTS
.
.
C100 20 FD AE   JSR $AEFD
C103 20 EB B7   JSR $B7EB
C106 84 5B      STY $5B
C108 85 5C      STA $5C
C10A A0 03      LDY #$03
```

```
C10C A5 14        LDA $14
C10E 85 58        STA $58
C110 A5 15        LDA $15
C112 85 59        STA $59
C114 86 5A        STX $5A
C116 18           CLC
C117 06 58        ASL $58
C119 26 59        ROL $59
C11B 88           DEY
C11C D0 F8        BNE $C116
C11E A9 B0        LDA #$B0
C120 18           CLC
C121 65 59        ADC $59
C123 85 59        STA $59
C125 C9 BF        CMP #$BF
C127 90 03        BCC $C12C
C129 4C 48 B2     JMP $B248
C12C A9 00        LDA #$00
C12E 85 57        STA $57
C130 A5 5A        LDA $5A
C132 A4 57        LDY $57
C134 91 58        STA ($58),Y
C136 A2 00        LDX #$00
C138 A4 5B        LDY $5B
C13A A5 5C        LDA $5C
C13C 20 F1 B7     JSR $B7F1
C13F 86 5A        STX $5A
C141 E6 57        INC $57
C143 A5 57        LDA $57
C145 C9 07        CMP #$07
C147 D0 E7        BNE $C130
C149 A5 5A        LDA $5A
C14B A4 57        LDY $57
C14D 91 58        STA ($58),Y
C14F 60           RTS
.
.
```

To go into user-defined character mode, enter SYS49152. This switches in bank 2 of memory, and copies all the character ROM into RAM. Having done this, you are left with about 29K of memory to play with, although one or two things have moved about a little. For instance, sprite data pointers now live at locations 33784 to 33791, sprite data must be stored from locations 33792 to 40959, and the screen memory map starts at location 32768 (remember that one, old Commodore owners?).

To then re-define a character, the second machine code routine must be used in the following way:

SYS 49408,A,1,2,3,4,5,6,7,8

where A is the character number to be re-defined, and the numbers 1 to 8 are the eight values for each row of the 8 by 8 matrix that makes up a character. For example:

SYS 49408,0,24,24,12,28,8,22,28,50

will re-define the '@' key to resemble a little man.

However, it is the approach rather than the program that interests us here. When writing musical programs in Basic, it soon becomes apparent that an awful lot of POKEs have to be performed before anything happens. How nice it would be to have one SYStem call that does everything: designates the voice, the attack, decay, sustain and release, the waveform, the pulse rate if necessary, and finally the low and high values of the note to be played.

The following machine code routine is the result of this idea:

```
B*
   PC   SR AC XR YR SP
.;8FEB 33 00 D3 00 F6
.
C000 20 FD AE    JSR $AEFD
C003 20 EB B7    JSR $B7EB
C006 84 5B       STY $5B
C008 85 5C       STA $5C
C00A A0 01       LDY #$01
C00C A5 14       LDA $14
C00E 85 58       STA $58
C010 A5 15       LDA $15
C012 85 59       STA $59
C014 86 5A       STX $5A
C016 8E 01 04    STX $0401
C019 A5 58       LDA $58
C01B 8D 00 04    STA $0400
C01E 88          DEY
C01F D0 F8       BNE $C019
C021 A9 B0       LDA #$B0
C023 18          CLC
C024 65 59       ADC $59
C026 85 59       STA $59
```

```
C028 C9 BF       CMP ##$BF
C02A 90 03       BCC $C02F
C02C 4C 48 B2    JMP $B248
C02F A9 00       LDA ##$00
C031 85 57       STA $57
C033 A5 5A       LDA $5A
C035 A4 57       LDY $57
C037 A2 00       LDX ##$00
C039 A4 5B       LDY $5B
C03B A5 5C       LDA $5C
C03D 20 F1 B7    JSR $B7F1
C040 A4 57       LDY $57
C042 86 5A       STX $5A
C044 A5 5A       LDA $5A
C046 99 02 04    STA $0402,Y
C049 E6 57       INC $57
C04B EA          NOP
C04C EA          NOP
C04D EA          NOP
C04E EA          NOP
C04F EA          NOP
C050 A5 57       LDA $57
C052 C9 06       CMP ##$06
C054 D0 DD       BNE $C033
C056 AD 00 04    LDA $0400
C059 C9 00       CMP ##$00
C05B F0 12       BEQ $C06F
C05D C9 01       CMP ##$01
C05F F0 07       BEQ $C068
C061 C9 02       CMP ##$02
C063 F0 08       BEQ $C06D
C065 4C 48 B2    JMP $B248
C068 69 05       ADC ##$05
C06A 4C 6F C0    JMP $C06F
C06D 69 0B       ADC ##$0B
C06F 8D 0A 04    STA $040A
C072 AE 0A 04    LDX $040A
C075 A9 00       LDA ##$00
C077 9D 04 D4    STA $D404,X
C07A AD 01 04    LDA $0401
C07D 9D 05 D4    STA $D405,X
C080 AD 02 04    LDA $0402
C083 9D 06 D4    STA $D406,X
C086 AD 03 04    LDA $0403
C089 9D 04 D4    STA $D404,X
C08C AD 04 04    LDA $0404
C08F 9D 02 D4    STA $D402,X
C092 AD 05 04    LDA $0405
C095 9D 03 D4    STA $D403,X
C098 AD 06 04    LDA $0406
C09B 9D 00 D4    STA $D400,X
C09E AD 07 04    LDA $0407
```

```
C0A1 9D 01 D4     STA $D401,X
C0A4 A9 0F        LDA #$0F
C0A6 9D 18 D4     STA $D418,X
C0A9 EA           NOP
C0AA 60           RTS
C0AB 00           BRK
.
.
```

Or, if you prefer Basic loaders:

```
COMMAND ADDER IN BASIC


10 FORI=0TO71:READA:B=B+A:POKE49152+I,A:NEXT
12 IFB<>8249THENPRINT"ERROR IN BLOCK #0":STOP
20 FORI=72TO143:READA:C=C+A:POKE49152+I,A:NEXT
22 IFC<>7728THENPRINT"ERROR IN BLOCK #1":STOP
30 FORI=144TO170:READA:D=D+A:POKE49152+I,A:NEXT
32 IFD<>2781THENPRINT"ERROR IN BLOCK #2":STOP
34 PRINT"DATA ENTERED CORRECTLY AND CODE NOW IN  P
LACE.":END
30000 REM BLOCK # 0
30001 DATA32,253,174,32,235,183,132,91
30002 DATA133,92,160,1,165,20,133,88
30003 DATA165,21,133,89,134,90,142,1
30004 DATA4,165,88,141,0,4,136,208
30005 DATA248,169,176,24,101,89,133,89
30006 DATA201,191,144,3,76,72,178,169
30007 DATA0,133,87,165,90,164,87,162
30008 DATA0,164,91,165,92,32,241,183
30009 DATA164,87,134,90,165,90,153,2
30010 REM BLOCK # 1
30011 DATA4,230,87,234,234,234,234,234
30012 DATA165,87,201,6,208,221,173,0
30013 DATA4,201,0,240,18,201,1,240
30014 DATA7,201,2,240,8,76,72,178
30015 DATA105,5,76,111,192,105,11,141
30016 DATA10,4,174,10,4,169,0,157
30017 DATA4,212,173,1,4,157,5,212
30018 DATA173,2,4,157,6,212,173,3
30019 DATA4,157,4,212,173,4,4,157
30020 REM BLOCK # 2
30021 DATA2,212,173,5,4,157,3,212
30022 DATA173,6,4,157,0,212,173,7
30023 DATA4,157,1,212,169,15,157,24
30024 DATA212,234,96

READY.
```

The command accepts the following syntax:

SYS49152,A,1,2,3,4,5,6,7

where A is the voice number (nought, one or two), 1 is the setting for the attack/decay cycle, 2 is the setting for the sustain/release cycle, 3 is the waveform, 4 is the low order setting for the pulse width if necessary and 5 is the high order, and 6 and 7 are the low and high order values for the frequency of the note to be played.

The following command:

SYS49152,0,9,0,65,255,0,73,4

will set everything and play the note middle C for voice 0. Any attempt to use a voice number greater than 2 will result in an error, and similarly any attempt to use a value for any of the other settings of greater than 255 will also result in an error.

```
TRAIN DEMO


10 FORI=10TO1STEP-1
20 SYS49152,0,9,0,129,0,0,10,10
30 FORK=1TO100*I
40 NEXTK,I
50 FORI=1TO50
60 SYS49152,0,9,0,129,0,0,10,10
70 FORK=0TO100:NEXTK,I

READY.



SIREN DEMO


10 FORI=100TO50STEP-1
20 SYS49152,0,9,0,65,10,255,10,I
30 FORK=1TO10
40 NEXTK,I
50 FORI=50TO100
60 SYS49152,0,9,0,65,10,255,10,I
70 FORK=0TO10:NEXTK,I
80 GOTO10

READY.
```

```
10 FORI=100TO50STEP-1
20 SYS49152,0,9,0,65,10,255,10,I
25 SYS49152,1,9,0,65,10,255,10,150-I
26 SYS49152,2,10,10,17,0,0,10,100
30 FORK=1TO10
40 NEXTK,I
50 FORI=50TO100
60 SYS49152,0,9,0,65,10,255,10,I
65 SYS49152,1,9,0,65,10,255,10,150-I
66 SYS49152,2,10,10,17,0,0,10,100
70 FORK=0TO10:NEXTK,I
80 GOTO10

READY.
```

How does all this work? Let's take the machine code listing. If you know about machine code, you'll be able to work it out for yourself, and if you don't, no explanation of how it works will be sufficient. So we're going to concentrate on how the program accepts the various parameters, how it determines how many parameters to accept, where to store them, and how to use them afterwards.

The routine has been written so that the parameters to be input after the SYStem call are stored on the screen. Type in a command, and you'll see them changing. The numbers 1,2,3, etc. in our earlier example are stored in the first seven screen locations. The code at $C016, $C01B, and $C046 takes care of this. Alter those values, and you can store the variables anywhere you like. Seven parameters are accepted after the initial number because of locations $C052 and $C053: change the 06, and you change the number of parameters to be input. Knowing where they are all stored, it is then a simple matter to retrieve those values and use them accordingly, which is what the code from $C072 onwards does.

## Datamaker

As a finale, you may be interested in this program, which turns the machine code programs into collections of data statements. That is all it does, since the early lines of each data program have to be added by hand, but it may save you some time.

```
100 INPUT"START ADDRESS";S:POKE830,S/256:INPUT"END
 ADDRESS";E:IFE<STHEN100
130 POKE828,0:POKE829,0:POKE831,0:Z=INT((E-S)/8):P
OKE829,Z
1450 IFPEEK(829)*8+7<PEEK(828)*8THEN1600
1451 S=PEEK(830):S=S*256+PEEK(831)*72
1455 PRINT"[CLR,3CD]";PEEK(828)+30000;"REM BLOCK #
";PEEK(828)/10
1460 POKE828,PEEK(828)+1:FORI=0TO8
1470 PRINTPEEK(828)+30000"DATA";
1480 FORJ=0TO7
1490 BB=PEEK(S+I*8+J)
1500 BB$=RIGHT$(STR$(BB),LEN(STR$(BB))-1)
1510 PRINTBB$;",";
1520 NEXT J
1530 PRINT"[CL] ";POKE828,PEEK(828)+1
1540 NEXT I
1560 PRINT"GOTO1450[HOME]"
1570 POKE 198,12:FORI=0TO11:POKE631+I,13:NEXTI:POK
E831,PEEK(831)+1:END
1600 PRINT"[CLR,3CD]";100:PRINT130:PRINT1450:PRINT
1451:PRINT1455:PRINT1460:PRINT1470:PRINT1480:PRINT
1490:PRINT1500:PRINT1510
1610 GOTO1700
1620 PRINT"[CLR,3CD]"1520:PRINT1530:PRINT1540:PRIN
T1560:PRINT1570:PRINT1600:PRINT1610:PRINT1620:PRIN
T1700:PRINT1710:PRINT1630
1630 GOTO1700
1700 PRINT"GOTO1620[HOME]"
1710 POKE 198,12:FORI=0TO11:POKE631+I,13:NEXTI:END
```

## Conclusion

These are just two ways of adding commands to Basic, and obviously there are more. However, these are two of the simplest, and using the techniques discussed it should be easy enough for you to produce your own additions to Basic.

Why use seven POKE commands when one SYStem call will do? Have fun with Basic.

# 15
# Technical Overview

This chapter is for anyone who wants to get the most out of the sound capabilities of the 64. We've added commands to Basic, we've intercepted Basic routines to give us continuous interrupt-driven music, and now, finally, comes the chance to see what the chip itself can do.

The 6581 is one of the most versatile and powerful of the musical chips currently available in any home computer, and can even compete with full blown musical synthesisers costing many times the price, as the following specifications show :

3 tone oscillators, or voices, each with a range of 0 to 4 KHz

4 waveforms per voice, covering triangle, sawtooth, variable pulse and noise.

3 amplitude modulators, with a range of 48 decibels.

3 envelope generators, featuring exponential response, an attack rate in the range 2ms to 8 seconds, decay rate in the range 6ms to 24s, sustain level from 0 up to peak volume, and a release rate from 6 ms to 24s.

Oscillator synchronization.

Ring modulation.

Programmable filtering, featuring a cut off range of 30 Hz to 12 KHz, a 12 decibel octave roll off, low pass, high pass, band pass and notch outputs, and a variable resonance.

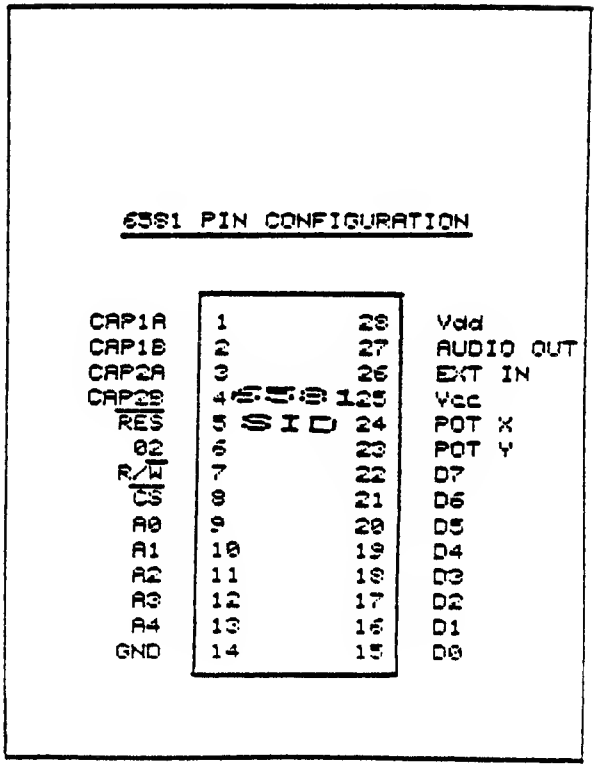
A master volume control (the one failing of this chip! It should have had separate volume controls)

2 A to D POT interfaces
A random number modulation generator, and

External audio output, allowing you to link the 6581 up to an external speaker perhaps, or play it through a guitar and then into a speaker, or even link up a couple of 6581s together.

# 6581 Pin Configuration

The 6581 pin configuration looks like this :



# More General Description

Here we'll outline in more human terms some of the capabilities of the 6581, and introduce a few terms that will be useful throughout the rest of this chapter.

The 6581 consists of three synthesiser voices, which can be used either independently or in conjunction with one another, in order to create some amazing sounds.

Each voice consists of a tone oscillator and waveform generator, an envelope generator and an amplitude modulator.

The tone oscillator controls the pitch of each voice, whilst each voice can also choose from one of four different waveforms at the tone, or pitch, selected, with complicated harmonic structure possible with each waveform.

The amplitude modulator, in conjunction with the envelope generator, produces the overall quality of the noise heard, and can be used to re-create the sounds of many musical instruments.

A programmable filter is also provided to enable complex tone colours to be produced: true synthesis on a home computer!

The 6581 allows the 6510 to read the changing output of the third oscillator and third envelope generator.

This can be used to create a variety of effects, including vibrato and frequency/filter sweeps. We've covered a few simple examples of this earlier.

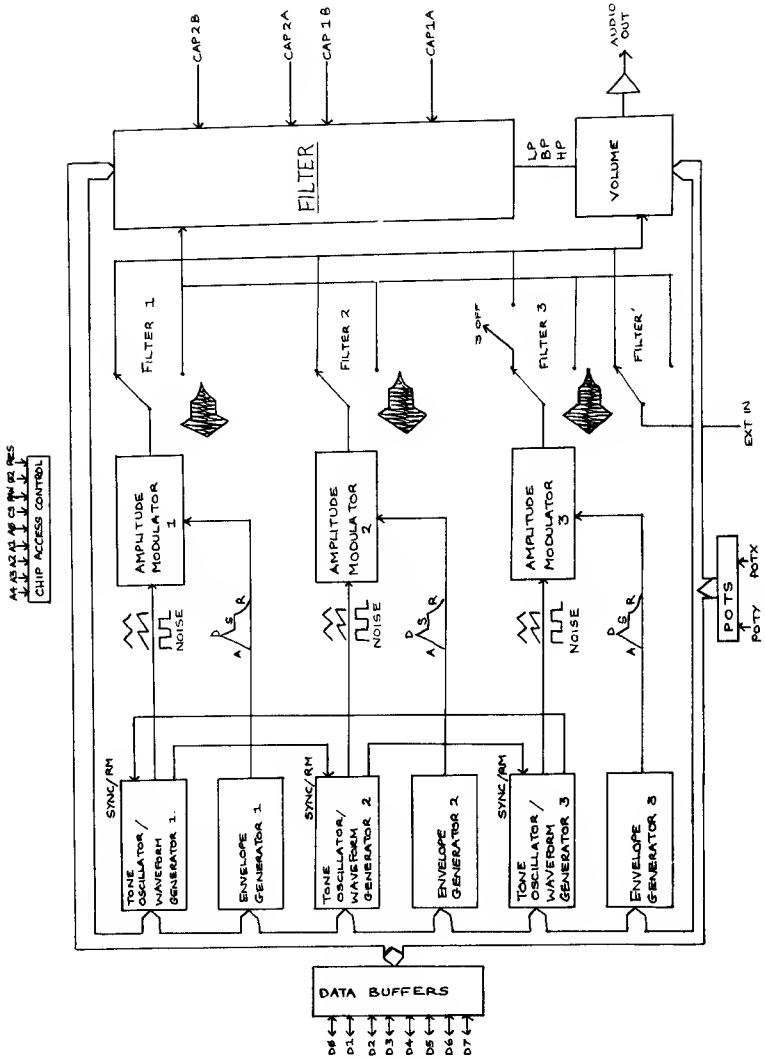
This third oscillator can also act as a random number generator.

The two A to D convertors are provided for linking the 6581 up to a number of potentiometers, for a variety of uses: perhaps as controls for some kind of external musical synthesiser.

In addition, we can link several 6581s up together, or mix them up into other external audio equipment.

The rest of this chapter goes into more detail on all of these features.

# 6581 Block Diagram



# 6581 Control Register

| | ADDRESS | | | | | REG # | DATA | | | | | | | | REG | REG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A4 | A3 | A2 | A1 | A0 | (HEX) | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | NAME | TYPE |
| | | | | | | | | | | | | | | | VOICE 1 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 00 | | | | | | | | | FREQ LO | WRITE-only |
| 1 | 0 | 0 | 0 | 0 | 1 | 01 | | | | | | | | | FREQ HI | WRITE-only |
| 2 | 0 | 0 | 0 | 1 | 0 | 02 | | | | | | | | | PW LO | WRITE-only |
| 3 | 0 | 0 | 0 | 1 | 1 | 03 | | | | | | | | | PW HI | WRITE-only |
| 4 | 0 | 0 | 1 | 0 | 0 | 04 | | | | | | | | | CONTROL REG | WRITE-only |
| 5 | 0 | 0 | 1 | 0 | 1 | 05 | | | | | | | | | ATTACK/DECAY | WRITE-only |
| 6 | 0 | 0 | 1 | 1 | 0 | 06 | | | | | | | | | SUSTAIN/RELEASE | WRITE-only |
| | | | | | | | | | | | | | | | VOICE 2 | |
| 7 | 0 | 0 | 1 | 1 | 1 | 07 | | | | | | | | | FREQ LO | WRITE-only |
| 8 | 0 | 1 | 0 | 0 | 0 | 08 | | | | | | | | | FREQ HI | WRITE-only |
| 9 | 0 | 1 | 0 | 0 | 1 | 09 | | | | | | | | | PW LO | WRITE-only |
| 10 | 0 | 1 | 0 | 1 | 0 | 0A | | | | | | | | | PW HI | WRITE-only |
| 11 | 0 | 1 | 0 | 1 | 1 | 0B | | | | | | | | | CONTROL REG | WRITE-only |
| 12 | 0 | 1 | 1 | 0 | 0 | 0C | | | | | | | | | ATTACK/DECAY | WRITE-only |
| 13 | 0 | 1 | 1 | 0 | 1 | 0D | | | | | | | | | SUSTAIN/RELEASE | WRITE-only |
| | | | | | | | | | | | | | | | VOICE 3 | |
| 14 | 0 | 1 | 1 | 1 | 0 | 0E | | | | | | | | | FREQ LO | WRITE-only |
| 15 | 0 | 1 | 1 | 1 | 1 | 0F | | | | | | | | | FREQ HI | WRITE-only |
| 16 | 1 | 0 | 0 | 0 | 0 | 10 | | | | | | | | | PW LO | WRITE-only |
| 17 | 1 | 0 | 0 | 0 | 1 | 11 | | | | | | | | | PW HI | WRITE-only |
| 18 | 1 | 0 | 0 | 1 | 0 | 12 | | | | | | | | | CONTROL REG | WRITE-only |
| 19 | 1 | 0 | 0 | 1 | 1 | 13 | | | | | | | | | ATTACK/DECAY | WRITE-only |
| 20 | 1 | 0 | 1 | 0 | 0 | 14 | | | | | | | | | SUSTAIN/RELEASE | WRITE-only |
| | | | | | | | | | | | | | | | FILTER | |
| 21 | 1 | 0 | 1 | 0 | 1 | 15 | | | | | | | | | FC LO | WRITE-only |
| 22 | 1 | 0 | 1 | 1 | 0 | 16 | | | | | | | | | FC HI | WRITE-only |
| 23 | 1 | 0 | 1 | 1 | 1 | 17 | | | | | | | | | RES/FILT | WRITE-only |
| 24 | 1 | 1 | 0 | 0 | 0 | 18 | | | | | | | | | MODE/VOL | WRITE-only |
| | | | | | | | | | | | | | | | MISC | |
| 25 | 1 | 1 | 0 | 0 | 1 | 19 | | | | | | | | | POTX | READ-only |
| 26 | 1 | 1 | 0 | 1 | 0 | 1A | | | | | | | | | POTY | READ-only |
| 27 | 1 | 1 | 0 | 1 | 1 | 1B | | | | | | | | | OSC3/RANDOM | READ-only |
| 28 | 1 | 1 | 1 | 0 | 0 | 1C | | | | | | | | | ENV3 | READ-only |

# 6581 Register Descriptions

We'll now spend a few pages going through each register in detail, using location 54272 as our base register, or register 0, starting with :

Voice 1 : Frequency Low/Frequency High
These two registers combine together to form a 16 bit number which linearly controls the frequency of voice 1.

This frequency is determined by the following equation :

Fout = (Fn *Fclk/16777216) Hz

where Fn is the 16 bit number in the frequency registers, and Fclk is the system clock applied to the 02 input, pin 6.

For our standard 1 MHz clock, this formula comes down to :

Fout = (Fn * 0.05961) Hz

Later on in this chapter we'll give you a much more complete table of musical notes, frequencies and values of Fn than were given earlier in the chapter on sound on the 64.

These values are all meant to be taken with a pinch of salt, and can be varied slightly as you, the listener, see fit.

It should also be noted here that the frequency resolution of the 6581 is such that sweeping from note to note on an even tempered scale is possible, without any noticeable frequency steps.

## Pulse Width Low and High

These two registers combine together to form a 12 bit number, which linearly controls the pulse width of the pulse waveform of voice one. Bits 4 to 7 of Pulse high are not used.

This pulse width is determined by the following equation :

PWout = (PWn/40.95) %

where PWn is the 12 bit number in the PW registers.

Again, the pulse width resolution is such that the width can be smoothly swept along without any noticeable stepping effects.

For constant pulse widths, a value of 0 or 4095 will produce a constant DC output, whilst a value of 2048 will produce a perfect square wave.

Obviously, these features cannot be used without having previously selected the pulse waveform for voice 1.

## Control Register

The most important register of them all, containing eight control bits with the following functions :

### Gate - Bit 0

This controls the envelope generator for voice 1, and when this bit

is set to a '1' the envelope generator is triggered and the Attack/Decay/Sustain (or ADS) cycle is begun.

When this bit is reset to a zero, then the release part of the cycle begins.

This envelope generator controls the amplitude of voice 1 as it appears at the audio output, and must therefore be triggered in order for the selected output of voice 1 to be audible.

**Sync - Bit 1**

When set to a '1' this synchronizes the fundamental frequency of voice 1 with the fundamental frequency of voice 3, producing what are known as 'hard sync' effects.

Varying the frequency of voice 1 with respect to voice 3 produces a wide range of complex harmonic structures from voice 1 at the frequency of voice 3.

In order for this to take place, obviously voice 3 must be set to some frequency or other, preferably lower than that of voice 1, but naturally higher than zero.

Nothing else connected with voice 3 has any effect on Sync.

**Ring Mod - Bit 2**

When set to a '1' this bit replaces the triangle waveform of voice 1 with a ring modulation combination of voices 1 and 3 : obviously, one must previously have selected the waveform of voice 1 to be a triangle one!

Varying the frequency of voice 1 with respect to voice 3 produces a wide range of non-harmonic overtone structures.

Again, nothing else connected with voice 3 has any effect on Ring Mod.

**Test - Bit 3**

This bit, when set to a '1', resets and holds voice 1 at zero, until the bit is cleared.

The noise waveform of voice 1 is also reset, and if a pulse wave has been selected this is held at a DC level.

Normally only used for testing purposes, and hence the name, it can be used to synchronize voice 1 to external events.

### Triangle - Bit 4

When this is set to a '1', the triangle waveform is selected for voice 1. This is low in harmonics, and thus produces a mellow, reed-like note.

### Sawtooth - Bit 5

When this is set to a '1', the sawtooth waveform is selected for voice 1. This is rich in harmonics, and thus produces a brassy, trumpet-like note.

### Pulse - Bit 6

When this is set to a '1' the pulse waveform is selected for voice 1. The harmonic content of this waveform can be varied by altering the pulse width registers, producing a wide variaty of different musical (and not so musical) sounds.

Sweeping through the pulse widths can produce some dynamic effects, and can add a sense of motion to the sound.

Rapidly altering from one pulse width to another can also be used to produce some interesting harmonic effects.

### Noise - Bit 7

When set to a '1', the noise waveform is selected for voice 1. This is a totally random signal which changes at the frequency of voice 1, and thus is of most use in generating purely sound 'effects', like missiles taking off, engines revving, or vast explosions.

The sound of waves lapping on the beach, or of a cymbal being rapidly hit, can be achieved by sweeping through the different frequencies.

One of the above waveforms must be selected in order for voice 1 to

produce any audible sound, although that sound can be turned off without un-selecting a waveform, as the voice at the end is a funcion of the envelope generator only.

Also, you cannot add more than one waveform together to produce something totally different from the above four.

You are welcome to try, but the most likely result is that voice 1 will be switched off, and can only be reset by the Test bit, or by setting pin 5 to low, or '0'.

## Attack/Decay

Bits 4 to 7 of this register, known as ATK0 to ATK3, select an attack rate from 0 to 15 for the voice 1 envelope generator. The Attack rate determines how fast the output of voice 1 rises from zero to peak amplitude, when the envelope generator is triggered.

Bits 0 to 3 of this register, known as DCY0 to DCY3, allow you to select a decay rate from 1 to 15 for the envelope generator. The decay cycle comes after the attack cycle, and determines how quickly the output falls from the peak amplitude to the selected sustain level.

## Sustain/Release

Bits 4 to 7 of this register, known as STN0 to STN3, allow you to select a sustain level from 0 to 15 for the envelope generator for voice 1. The sustain cycle follows the decay cycle, and determines at what amplitude voice 1 will remain as long as the trigger bit remains set. This is all done on a linear basis, so, for example, a sustain level of 8 would cause voice 1 to sustain at exactly half the peak amplitude reached by the attack cycle.

Bits 0 to 3 of this register, known as RLS0 to RLS3, allow you to select a release rate from 0 to 15 for the envelope generator of voice 1. The release cycle follows the sustain cycle, and determines how rapidly the amplitude of voice 1 will fall from the sustain level to zero amplitude.

The 16 release rates are identical to the decay rates, shown below.

# Envelope Rates

The cycling of this envelope generator can be altered at any point in the cycle by the gate bit, as the generator can be gated and released at any time, without restriction.

So, if the gate bit is set whilst half way through an attack cycle, the release cycle will begin immediately, and if the gate is reset again whilst the release cycle is still continuing, another attack cycle will start from whatever amplitude had been reached during release.

As you might imagine, this gets a bit hairy after a while, but does allow quite complex effects to be achieved.

**Envelope Rates**

| Value Dec | Value Hex | Attack Rate (Time/Cycle) ms | Decay/Release Rate (Time/Cycle) ms |
|---|---|---|---|
| 0 | 0 | 2 | 6 |
| 1 | 1 | 8 | 24 |
| 2 | 2 | 16 | 48 |
| 3 | 3 | 24 | 72 |
| 4 | 4 | 38 | 114 |
| 5 | 5 | 56 | 168 |
| 6 | 6 | 68 | 204 |
| 7 | 7 | 80 | 240 |
| 8 | 8 | 100 | 300 |
| 9 | 9 | 250 | 750 |
| 10 | A | 500 | 1.5 sec |
| 11 | B | 800 | 2.4 sec |
| 12 | C | 1 sec | 3 sec |
| 13 | D | 3 sec | 9 sec |
| 14 | E | 5 sec | 15 sec |
| 15 | F | 8 sec | 24 sec |

# Voice 2 and 3

### Voice 2

The registers $07 to $0D control voice 2, and function in the same way as registers $00 to $06 for voice 1, with the following two exceptions :

1) When SYNC is selected, it synchronizes voice 2 with voice 1.
2) When RING MOD is selected, it replaces the triangle output of voice 2 with the ring modulated combination of voices 2 and 1.

### Voice 3

The registers $0E to $14 control voice 3, and function in the same way as registers $00 to $06 for voice 1, with the following two exceptions :

1) When SYNC is selected, it synchronizes voice 3 with voice 2.
2) When RING MOD is selected, it replaces the triangle output of voice 3 with the ring modulated combination of voices 3 and 2.

# Filtering

### Freq Lo/Freq Hi - registers $15 and $16.

As bits 3 to 7 of register $15 are not used, these two combine together to form an 11 bit number which linearly controls the cutoff, or centre frequency of the programmable filter. The approximate cutoff frequency is obtained from :

FCout = ((6.6E-8 + FCn * 1.28E-8)/C) Hz

where FCn is the 11 bit number in the above two registers and C is the value of the two filter capacitors connected to pins 1 to 4, or in our case 2200 picoFarads.

This gives an approximate filter range of 30 Hz to 12 KHz, according to :

FCout = (30 + FCn * 5.8) Hz

This frequency response can be altered for specific applications, but I don't recommend you doing it!

## Res/Filt - Register $17

Bits 4 to 7 of this register control the resonance of the filter, where resonance emphasises components of the frequency at the cutoff frequency of the filter, thus causing a sharper sound.

There are 16 resonance settings ranging linearly from no resonance, when this is set to zero, or maximum resonance, when it is set to 15.

Bits 0 to 3 determine which signals will be routed through the filter :

Bit 0: When this is set to zero, voice 1 appears directly at the audio output, and there is no filtering effect. When set to 1, voice 1 is processed through the filter, and the harmonic content of voice 1 is altered according to the selected filter parameters.

Bit 1: Ditto for voice 2

Bit 2: Ditto for voice 3

Bit 3: Ditto for external audio input on pin 26.

## Mode/Vol - Register $18

We've already seen this one as the master volume control, but it actually does a whole lot more.

Bits 0 to 3 are the actual volume settings, and allow you to select an overall volume ranging from 0 (silence) to 15 (maximum).

Bits 4 through 7 select various filter modes and output options :

Bit 4 - When set to a '1', the low pass output of the filter is selected and sent to the audio output. For a given filter input signal, all components of the frequency below the filter cut off are passed

through unaltered, whilst all those above the cutoff are attenuated at a rate of 12 decibels per octave.

Bit 5 - As above for band pass output, but attenuation above and below the cutoff is at a rate of 6 decibels per octave.

Bit 6 - As above for high pass output, and attenuation below the cutoff is back to 12 decibels per octave.

Bit 7 - When this is set to '1' the output of voice 3 is disconnected from the direct audio path, so setting voice 3 to bypass the filter and setting 3 OFF to a '1' stops voice 3 from reaching the audio output. Thus voice 3 can be used for modulation purposes without any extraneous noises coming out.

These filter modes are additive, in that one can combine a number of different modes at the same time. Playing with, and understanding, these frequency alterations is the key to getting the most out of the 6581.

## Miscellaneous Information

POTX - Register $19

This allows the processor to read the position of the potentiometer connected to POTX, or pin 24, with a range of 0 at minimum resistance to 255 at maximum resistance. This value is constantly being updated.

POTY - Register $1A

As above, for POTY, at pin 23.

OSC 3/Random - Register $1B

This allows the processor to read the upper 8 output bits of oscillator three, and it is worth experimenting to read the numbers generated when producing the various waveforms. Noise produces a series of random numbers, and thus this can also be used as a random number generator in preference to RND.

Its chief purpose is to act as a modulation generator, by combining the numbers produced with something like the filter frequency registers, or the pulse width, to produce a virtually unlimited range of effects.

Voice 3 should be set to zero when using this mode.

ENV 3 - Register $1C

As above, but for the voice 3 envelope generator, and this is usually used in conjunction with the frequency filter.

The voice 3 envelope generator must be gated in order to produce any effects from this process.

# 6581 Pin Description

### CAP1A,CAP1B (Pins 1&2)/CAP2A,CAP2B (Pins 3&4)

These four pins are used to connect the two integrating capacitors required by the programmable filter. C1 connects pins 1 and 2, and C2 connects pins 3 and 4.

The maximum cutoff frequency FCmax is given by :

FCmax = 2.6E-5 / C

where C is the capacitor value, in our case equal to 2200 pico farads.

### RES - Pin 5

This is the reset control for the 6581, and is connected to the reset line of the 6510.

### 02 - Pin 6.

This is the master clock for the 6581, and all oscillation frequencies and envelope rates are referenced from this clock. It also controls data transfer from the 6581 to the 6510,

### R/W - Pin 7

This controls the direction of data transfer between the 6581 to the 6510. When this line is set to a '1' the 6510 can read from the selected 6581 register, and when set to a '0' the 6510 can write to the selected 6581 register. Thus this pin is connected to the system read/write line.

Note the read/write capabilities of the various registers as described earlier.

**CS - Pin 8**

This is a low-active chip select which controls data transfer between the 6581 and the 6510. Data transfer can only take place when this is set to low, and indeed when R/W is high, we get the 6510 reading from the 6581, and when R/W is low we get the 6510 transferring data to the 6581.

**A0 - A4 - Pins 9 to 13**

These inputs are used to select one of the 29 registers in the 6581. Sharp-eyed readers will observe that we have in fact the capacity to select any one of 32 registers. However, three are not used (perhaps they were once intended to be seperate volume controls ?). PEEKing these registers will only return a 0.

**GND - Pin 14**

This is simply the ground line.

**D0 - D7 - Pins 15 to 22.**

These 8 bi-directional lines are used in the transfer of data between the 6581 and the 6510. When writing data, the data buffers remain in the off state, and in read mode they are on, when the 6581 supplies data to the 6510.

**POTY, POTX - Pins 23 and 24**

These are the inputs to the A/D convertors used to digitize the position of any potentiometers connected up to the system.

**Vcc - Pin 25**

This is just used to minimise noise, and is linked up to the power supply with a 5 volt line.

### EXT IN - Pin 26

This analog input allows external audio signals to be mixed with the audio output of the 6581, or processed through the filter. This can be fed to mix outputs of multiple 6581s by daisy chaining them. The maximum number of chips that can be linked together in this way is limited only by the amount of noise and distortion allowable at the final output.

### AUDIO OUT - Pin 27

This is the line that carries the final audio output of the 6581, including the voices, filtering, and any external output. The output level of all the output is selected by the master volume control, register 24.

### Vdd - Pin 28

Again this is there to minimise noise, and requires a 12V line.

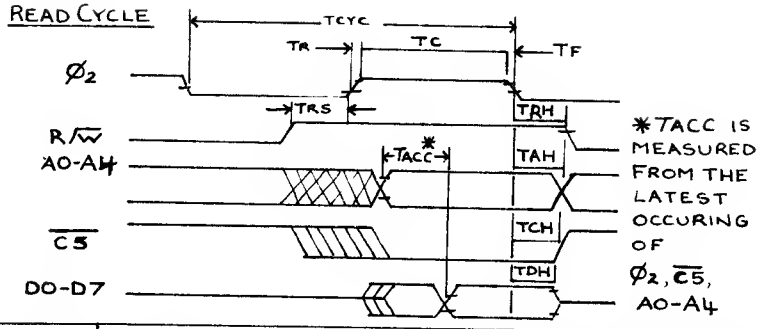
## 6581 Characteristics

ABSOLUTE MAXIMUM RATINGS

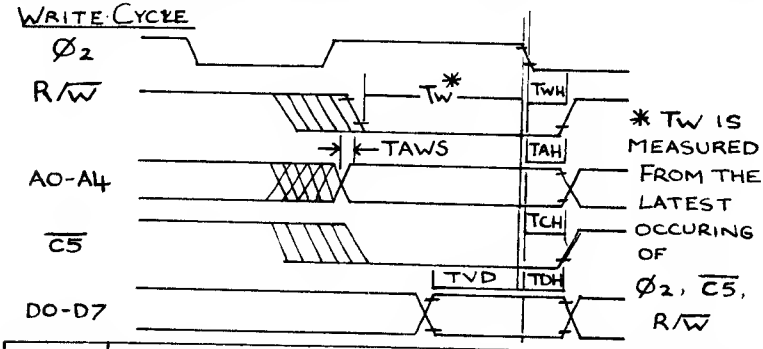
| RATING | SYMBOL | VALUE | UNITS |
|---|---|---|---|
| Supply Voltage | Vdd | -0.3 to +17 | VDC |
| Supply Voltage | Vcc | -0.3 to +7 | VDC |
| Input Voltage (analog) | Vina | -0.3 to +17 | VDC |
| Input Voltage (digital) | Vind | -0.3 to +7 | VDC |
| Operating Temperature | Ta | 0 to +70 | °C |
| Storage Temperature | Tstg | -55 to +150 | °C |

| ELECTRICAL CHARACTERISTICS (Vdd=12 ± 5% VDC, Vcc=5 ± 5% VDC, Ta=0 to 70°C) | | | | | | |
|---|---|---|---|---|---|---|
| CHARACTERISTIC | | SYMBOL | MIN | TYP | MAX | UNITS |
| Input High Voltage | (RES, Ø2, R/W, CS, A0-A4, D0-D7) | Vih | 2 | - | Vcc | VDC |
| Input Low Voltage | | Vil | -0.3 | - | 0.8 | VDC |
| Input Leakage Current | (RES, Ø2, R/W, CS, A0-A4; Vin=0-5 VDC) | Iin | - | - | 2.5 | uA |
| Three-State (Off) Input Leakage Current | (D0-D7; Vcc=max, Vin=0.4-2.4 VDC) | Itsi | - | - | 10 | uA |
| Output High Voltage | (D0-D7; Vcc=min, Iload=200 uA) | Voh | 2.4 | - | Vcc-0.7 | VDC |
| Output Low Voltage | (D0-D7; Vcc=max, Iload=3.2 mA) | Vol | GND | - | 0.4 | VDC |
| Output High Current | (D0-D7; Sourcing, Voh=2.4 VDC) | Ioh | 200 | - | - | uA |
| Output Low Current | (D0-D7; Sinking, Vol=0.4 VDC) | Iol | 3.2 | - | - | mA |
| Input Capacitance | (RES, Ø2, R/W, CS, A0-A4, D0-D7) | Cin | - | - | 10 | pF |
| Pot Trigger Voltage | (POTX, POTY) | Vpot | - | Vcc/2 | - | VDC |
| Pot Sink Current | (POTX, POTY) | Ipot | 500 | - | - | uA |
| Input Impedance | (EXT IN) | Rin | 100 | 150 | - | KOhms |
| Audio Input Voltage | (EXT IN) | Vin | 5.7<br>- | 6<br>0.5 | 6.3<br>3 | VDC<br>VAC |
| Audio Output Voltage | (AUDIO OUT, 1 KOhm load, volume=max)<br>One Voice on:<br>all Voices on: | Vout | 5.7<br>0.4<br>1.0 | 6<br>0.5<br>1.5 | 6.3<br>0.6<br>2.0 | VDC<br>VAC<br>VAC |
| Power Supply Current | (Vdd) | Idd | - | 20 | 35 | mA |
| Power Supply Current | (Vcc) | Icc | - | 70 | 100 | mA |
| Power Dissipation | (Ta=0) | Pd | - | 600 | 1000 | mW |

# 6581 Timing




| SYMBOL | NAME | MIN | TYP | MAX | UNITS |
|---|---|---|---|---|---|
| TCYC | Clock Cycle Time | 1 | – | 20 | µS |
| TC | Clock High Pulse Width | 450 | 500 | 10,000 | nS |
| TR, TF | Clock Rise/Fall Time | – | – | 25 | nS |
| TRS | Read Set-up Time | 0 | – | – | nS |
| TRH | Read Hold Time | 0 | – | – | nS |
| TACC | Access Time | – | – | 300 | nS |
| TAH | Address Hold Time | 10 | – | – | nS |
| TCH | Chip Select Hold Time | 0 | – | – | nS |
| TDH | Data Hold Time | 20 | – | – | nS |




| SYMBOL | NAME | MIN | TYP | MAX | UNITS |
|---|---|---|---|---|---|
| TW | Write Pulse Width | 350 | – | – | nS |
| TWH | Write Hold Time | 0 | – | – | nS |
| TAWS | Address Set-up Time | 0 | – | – | nS |
| TAH | Address Hold Time | 10 | – | – | nS |
| TCH | Chip Select Hold Time | 0 | – | – | nS |
| TVD | Valid Data | 80 | – | – | nS |
| TDH | Data Hold Time | 10 | – | – | nS |

# Equal Tempered Musical Scale Values

| MUSICAL NOTE | | FREQ (Hz) | OSC Fn (DECIMAL) | OSC Fn (HEX) | MUSICAL NOTE | | FREQ (Hz) | OSC Fn (DECIMAL) | OSC Fn (HEX) |
|---|---|---|---|---|---|---|---|---|---|
| 0 | C0 | 16.35 | 274 | 0112 | 48 | C4 | 261.63 | 4389 | 1125 |
| 1 | C0# | 17.32 | 291 | 0123 | 49 | C4# | 277.18 | 4650 | 122A |
| 2 | D0 | 18.35 | 308 | 0134 | 50 | D4 | 293.66 | 4927 | 133F |
| 3 | D0# | 19.44 | 326 | 0146 | 51 | D4# | 311.13 | 5220 | 1464 |
| 4 | E0 | 20.60 | 346 | 015A | 52 | E4 | 329.63 | 5530 | 159A |
| 5 | F0 | 21.83 | 366 | 016E | 53 | F4 | 349.23 | 5859 | 16E3 |
| 6 | F0# | 23.12 | 388 | 0184 | 54 | F4# | 370.00 | 6207 | 183F |
| 7 | G0 | 24.50 | 411 | 018B | 55 | G4 | 392.00 | 6577 | 19B1 |
| 8 | G0# | 25.96 | 435 | 0183 | 56 | G4# | 415.30 | 6968 | 1B38 |
| 9 | A0 | 27.50 | 461 | 01CD | 57 | A4 | 440.00 | 7382 | 1CD6 |
| 10 | A0# | 29.14 | 489 | 01E9 | 58 | A4# | 466.16 | 7821 | 1E8D |
| 11 | B0 | 30.87 | 518 | 0206 | 59 | B4 | 493.88 | 8286 | 205E |
| 12 | C1 | 32.70 | 549 | 0225 | 60 | C5 | 523.25 | 8779 | 2248 |
| 13 | C1# | 34.65 | 581 | 0245 | 61 | C5# | 554.37 | 9301 | 2455 |
| 14 | D1 | 36.71 | 616 | 0268 | 62 | D5 | 587.33 | 9854 | 267E |
| 15 | D1# | 38.89 | 652 | 028C | 63 | D5# | 622.25 | 10440 | 28C8 |
| 16 | E1 | 41.20 | 691 | 0283 | 64 | E5 | 659.26 | 11060 | 2B34 |
| 17 | F1 | 43.65 | 732 | 02DC | 65 | F5 | 698.46 | 11718 | 2DC6 |
| 18 | F1# | 46.25 | 776 | 0308 | 66 | F5# | 740.00 | 12415 | 307F |
| 19 | G1 | 49.00 | 822 | 0336 | 67 | G5 | 783.99 | 13153 | 3361 |
| 20 | G1# | 51.91 | 871 | 0367 | 68 | G5# | 830.61 | 13935 | 366F |
| 21 | A1 | 55.00 | 923 | 0398 | 69 | A5 | 880.00 | 14764 | 39AC |
| 22 | A1# | 58.27 | 978 | 03D2 | 70 | A5# | 932.33 | 15642 | 3D1A |
| 23 | B1 | 61.74 | 1036 | 040C | 71 | B5 | 987.77 | 16572 | 408C |
| 24 | C2 | 65.41 | 1097 | 0449 | 72 | C6 | 1046.50 | 17557 | 4495 |
| 25 | C2# | 69.30 | 1163 | 0488 | 73 | C6# | 1108.73 | 18601 | 48A9 |
| 26 | D2 | 73.42 | 1232 | 04D0 | 74 | D6 | 1174.66 | 19708 | 4CFC |
| 27 | D2# | 77.78 | 1305 | 0519 | 75 | D6# | 1244.51 | 20897 | 518F |
| 28 | E2 | 82.41 | 1383 | 0567 | 76 | E6 | 1318.51 | 22121 | 5669 |
| 29 | F2 | 87.31 | 1465 | 0589 | 77 | F6 | 1396.91 | 23436 | 588C |
| 30 | F2# | 92.50 | 1552 | 0610 | 78 | F6# | 1479.98 | 24830 | 60FE |
| 31 | G2 | 98.00 | 1644 | 066C | 79 | G6 | 1567.98 | 26306 | 66C2 |
| 32 | G2# | 103.83 | 1742 | 06CE | 80 | G6# | 1661.22 | 27871 | 6CDF |
| 33 | A2 | 110.00 | 1845 | 0735 | 81 | A6 | 1760.00 | 29528 | 7358 |
| 34 | A2# | 116.54 | 1955 | 07A3 | 82 | A6# | 1864.65 | 31284 | 7A34 |
| 35 | B2 | 123.47 | 2071 | 0817 | 83 | B6 | 1975.53 | 33144 | 8178 |
| 36 | C3 | 130.81 | 2195 | 0893 | 84 | C7 | 2093.00 | 35115 | 8928 |
| 37 | C3# | 138.59 | 2325 | 0915 | 85 | C7# | 2217.46 | 37203 | 9153 |
| 38 | D3 | 146.83 | 2463 | 099F | 86 | D7 | 2349.32 | 39415 | 99F7 |
| 39 | D3# | 155.56 | 2610 | 0A32 | 87 | D7# | 2489.01 | 41759 | A31F |
| 40 | E3 | 164.81 | 2765 | 0ACD | 88 | E7 | 2637.02 | 44242 | ACD2 |
| 41 | F3 | 174.61 | 2930 | 0B72 | 89 | F7 | 2793.83 | 46873 | B719 |
| 42 | F3# | 185.00 | 3104 | 0C20 | 90 | F7# | 2959.95 | 49660 | C1FC |
| 43 | G3 | 196.00 | 3288 | 0CD8 | 91 | G7 | 3135.96 | 52613 | CD85 |
| 44 | G3# | 207.65 | 3484 | 0D9C | 92 | G7# | 3322.44 | 55741 | D9BD |
| 45 | A3 | 220.00 | 3691 | 0E68 | 93 | A7 | 3520.00 | 59056 | E680 |
| 46 | A3# | 233.08 | 3910 | 0F46 | 94 | A7# | 3729.31 | 62567 | F467 |
| 47 | B3 | 246.94 | 4143 | 102F | 95 | B7 | 3951.06 | *66288 | *1F2F0 |

The table above provides a simple and quick method for generating the equal tempered scale. However, it is not very efficient on memory, since it requires 192 bytes just for the table. It would be better to determine each note algorithmically, using the fact that each note in an octave is half the frequency of that note in the next octave, which would bring us down to a table entry of just 12 entries rather than 92 as featured here.

Having done this, we could then specify each note by a single byte. Four bits could specify which of the 12 notes we wish to play in the octave (tones and semi-tones), and three bits to specify which octave

we wish to play in. In other words, use two separate nibbles for each note.

## 6581 Envelope Generators

The four-part envelope generator used in the 6581, the Attack Decay Sustain Release (ASDR) one, is one of the easiest of the generators to program, as well as giving some of the best results of any electronically generated musical sound.

By appropriate selection of the various envelope parameters the simulation of many popular musical instruments is possible, particularly of percussion instruments, or those where the sound is sustained for an appreciable period.

For instance, the piano. This immediately reaches full volume as soon as the key is struck, and then begins to die away again equally quickly. As long as the key is held down the decay rate is rather slow, but if the key is released, then decay is immediate.

The ADSR diagram might look something like this :

Attack : 02ms
Decay: 9 750ms
Sustain : 0
Release : 06ms

A percussion instrument, such as a cymbal, is characterised again by an immediate attack, but no sustain period as it gradually decays away to silence, and the parameters for that might be something like : =

Attack : 02ms
Decay: 9 750ms
Sustain : 0
Release : 9 750ms

Wind instruments, such as flutes and oboes, have a characteristically slow rise to maximum volume, then an intermediate sustain for as long as someone keeps supplying the air, and then a slow decay, perhaps like this :

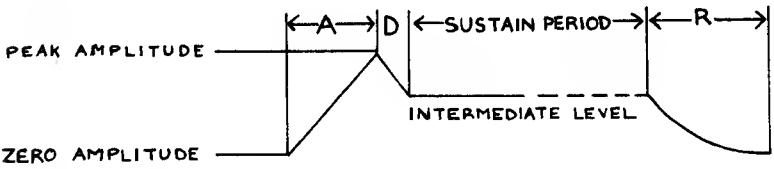
Attack : 10 500ms
Decay: 8 300ms
Sustain : 10
Release : 9 750ms

The simplest one of all would be the organ, which reaches and remains at full volume whilst the key is pressed and until it is released again, whence it drops immediately to zero volume, rather like this : =

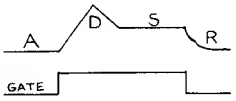
Attack : 02ms
Decay: 06ms
Sustain : 15
Release : 06ms

Of course, the 6581 is capable of much more than just mere impersonation, and the many unique sounds that can be generate by it can only really be discovered by a lot of experimentation.

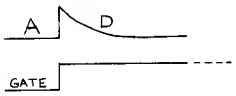
However, a good knowledge of envelope generation will certainly smooth the path, and playing about with ring modulation and filtering can again only make life a lot easier.



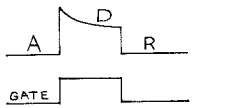

| | | |
|---|---|---|
| ATTACK : | 10 ($A) | 500 ms |
| DECAY : | 8 | 300 ms |
| SUSTAIN : | 10 ($A) | |
| RELEASE : | 9 | 750 ms |



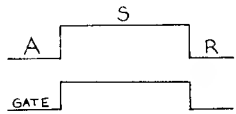

| | | |
|---|---|---|
| ATTACK : | 0 | 2 ms |
| DECAY : | 9 | 750 ms |
| SUSTAIN : | 0 | |
| RELEASE : | 9 | 750 ms |



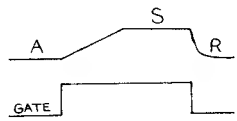

| | | |
|---|---|---|
| ATTACK : | 0 | 2 ms |
| DECAY : | 9 | 750 ms |
| SUSTAIN : | 0 | |
| RELEASE : | 0 | 6 ms |



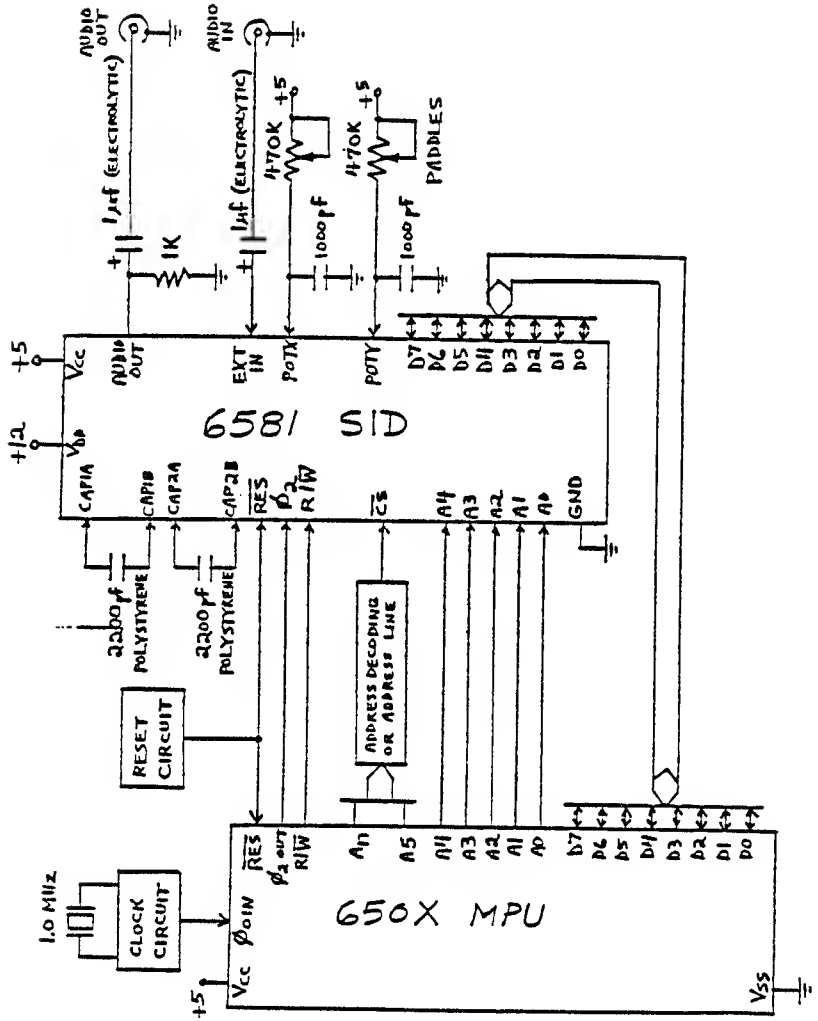

| | | |
|---|---|---|
| ATTACK : | 0 | 2 ms |
| DECAY : | 0 | 6 ms |
| SUSTAIN : | 15 ($F) | |
| RELEASE : | 0 | 6 ms |




| | | |
|---|---|---|
| ATTACK : | 10 ($A) | 500 ms |
| DECAY : | 0 | 6 ms |
| SUSTAIN : | 15 ($F) | |
| RELEASE : | 3 | 72 ms |



# Typical 6581 Application



# Index


6581 block diagram : 167
6581 characteristics : 179
6581 control register : 168
6581 envelope generators : 182
6581 pin configuration : 165
6581 pin description : 177
6581 register descriptions : 168
6581 timing : 180
Adding commands to Basic : 141
ADSR map: 33, 34
ADSR settings : 24
Basic commands, addition of : 141
Brass instruments : 16
Butterfield music : 43
Character alteration : 154
Character Get routine : 145
Charget, altering it : 149
Charset program : 86
Datamaker program : 162
Envelope generation : 34
Envelope rates : 24
Filtering : 35, 84
Frequency : 20,30
Harmonics : 31
Harmony : 42
Interrupts : 130
Keyboard instruments : 17, 18
Listing conventions : 12, 13
Machine code and music : 130
Memory saver program : 64
Music driver program : 138
Musical games : 115
Musical Hangman program : 46
Musical instruments : 15, 93
Musical interrupts : 130
Musical notation : 18, 19
Musical Practice program : 68
Musical Simon program : 56
Musical tables : 21
Musicman program : 119
Noise : 39
Note playing : 37
Note recognition program : 76
Percussion instruments : 17
Pestroy program : 115
Physics of sound : 20
Pulse width : 32, 33
Ring modulation : 67
Sample tunes : 124
Scale Practice program : 105
Sound effects : 104
String instruments : 15, 16
Synchronisation : 67
Synthesiser program : 93
System calls : 158
SID memory map : 25, 26
Technical overview : 164
Waveforms : 28
Woodwind instruments : 16, 17